UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FIN DE GRADO

ANÁLISIS DE FUNCIONES FÍSICAS INCLONABLES (PUFS) EN FPGA

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: Ayoub El Maataoui Nahal

Tutor: Honorio Martín González

Leganés, 21 de Junio de 2017

Agradecimientos

Me gustaría expresar mi gratitud a mi tutor Honorio por guiarme y darme soporte siempre que lo he necesitado a lo largo de la realización de este trabajo. También me gustaría agradecer a mi familia y amigos por darme su apoyo y ánimos para llegar a donde estoy.

Resumen

En la actualidad las FPGAs cobran cada vez más importancia y cubren un amplio rango de aplicaciones en sectores como la automoción, aeroespacial, defensa o medicina. Si la FPGA no dispone de medidas de seguridad efectivas para proteger la información albergada o la propia información de la FPGA, esta información puede se sustraída para diferentes fines. Si la información de la FPGA es sustraída puede ser clonada para ser comercializada de forma irregular. La clonación de circuitos integrados provoca grandes pérdidas, porque a pesar de que el clon es igual al circuito integrado clonado no cumple los mismos estándares de calidad. Por tanto tienen un ciclo de vida más corto y pueden provocar fallos de funcionamiento en los sistemas en los cuales estén integrados.

Esto hechos provocan que para enfrentar estos problemas de seguridad sean necesarias soluciones cada más fiable y eficientes. Una de estas soluciones es la utilización de funciones física inclonables (PUFs). Las funciones físicas inclonables es un concepto reciente, pero que cada vez es más aplicado en diferentes aplicaciones. Este concepto se basa en variaciones aleatorias que surgen durante el proceso de fabricación, y que causan que las propiedades físicas de uno dispositivo sean únicas, estas variaciones se utilizan para generar claves únicas e impredecibles que pueden ser utilizadas en diversas aplicaciones como la identificación o autenticación de dispositivos. Estas claves són únicas e impredecibles porque las variaciones de las propiedades físicas son aleatorias y no pueden ser controladas desde el exterior. En el presente trabajo se estudiarás y analizará la aplicación de funciones físicas inclonables en FPGAs para ver como se ven afectadas por diversos factores.

Abstract

FPGAs are becoming increasingly important and cover a wide range of applications in areas such as automotive, aerospace, defense or medicine. If the FPGA does not include effective security measures to protect the information hosted or the FPGA information itself, this information can be subtracted for different purposes. If the FPGA information is subtracted it can be cloned to be marketed irregularly. The cloning of integrated circuits causes large losses because even though the clone is equal to the cloned integrated circuit it does not meet the same quality standards. Hence, they have a shorter life cycle and can cause malfunctions in the systems in which they are integrated.

This fact proves that in order to face these security problems are necessary more reliable and efficient solutions. One of these solutions is the use of physical unclonable functions (PUFs). The physical unclonable functions are a recent concept, but that is increasingly applied in different applications. This concept is based on random variations that arise during the manufacturing process, and which causes each device to posses unique physical properties, these variations are used to generate unique and unpredictable keys that can be used in various applications such as identification or authentication of devices. These keys are unique and unpredictable because physical properties variations are random and cannot be controlled from the outside. In the present work, we will study and analyze the application of physical unclonable functions on FPGAs to see how they are affected by several factors.

Índice

Índice de Figuras

Índice de Tablas

Listado de Acrónimos

CLB	Configurable Logic Block
DCM	Digital Clock Manager
FPGA	Field Programmable Gate Array
I/O	Input/Output
IC	Integrated Circuit
ICON	Integrated Controller
ILA	Integrated Logic Analyzer
IP	Intellectual Property
LISA	Longest Increasing Subsequence-Based Grouping Algoritm
NVM	Non-Volatile Memory
POWF	Physical One-Way Function
PRF	Physical Random Function
PUF	Physical Unclonable Function
RAM	Random Acces Memory
RO	Ring Oscillator
SRAM	Static Random-Acces Memory
UCF	User Constraint File
VHDL	VHSIC Hardware Description Language
VIO	Virtual Input/Output

1 Introducción

1.1 Motivación

La globalización y la evolución de las tecnologías de información, provoca la utilización de volúmenes de información cada vez más grandes. En muchos casos, estos volúmenes contienen información sensible, y por tanto se convierten en objetivos de ataques para sustraer dicha información. Para evitar estos ataques, los volúmenes de información deben ser identificados y protegidos de forma segura. Por estos motivos cada vez cobra más relevancia el desarrollo de tecnologías, que ayuden a solventar problemas de seguridad e identificación de forma eficiente.

Algunas de estas tecnologías están destinadas al uso en FPGAs, que en la actualidad tienen un rango de aplicación muy amplio. Podemos encontrar aplicaciones en campos como la comunicación, procesamiento de datos, militar, aeroespacial o automoción entre otros. Al tener tantas aplicaciones, la protección de la información guardada, tanto de la aplicación ejecutada, como de la propia FPGA, puede llegar a ser un factor crítico, ya que si la información es sustraída se puede copiar la aplicación o incluso clonar la FPGA para ser comercializada de forma irregular, generándose grandes pérdidas. Según la empresa de análisis críticos IHS, la falsificación y reutilización de circuitos integrados puede suponer unas pérdidas potenciales para la industria electrónica de 169 billones de dólares cada año [19].

El principal problema de la falsificación de chips es que, aunque un chip falso sea exacto al original, no ha superado las pruebas de calidad necesarias y por tanto tienen ciclos de vida más cortos y pueden provocar fallos en los sistemas en lo que se utilizan, esto puede conllevar resultados catastróficos, si el chip es utilizado en sectores como la medicina, automoción o el aeroespacial entre otros.

Para evitar los problemas citados, normalmente se generan claves secretas que se almacenan en memorias no volátiles (NVM), pero esta solución puede llegar a ser difícil de implementar y costosa. Para proporcionar una alta seguridad los dispositivos electrónicos deben estar protegidos con costosos circuitos que necesitan una alimentación continua para detectar si el dispositivo es manipulado.

Una alternativa cada vez más utilizada a esta solución es la utilización de funciones físicas inclonables, conocidas por su acrónimo en inglés PUF, para la autenticación e identificación, combatir la falsificación de circuitos integrados (ICs) o la generación y protección de claves

criptográficas. PUF es una función que se basa en las propiedades físicas, únicas para cada dispositivo. Explota las diferencias entre los componentes físicos, que surgen durante el proceso de fabricación, para generar salidas impredecibles. Estas diferencias no pueden ser controladas desde el exterior porque son el resultado de influencias aleatorias e incontrolables. Por lo tanto, es extremadamente complicado o imposible producir dos ICs idénticos con las mismas propiedades físicas.

Al igual que las huellas dactilares se utilizan para identificar a personas, con las PUFs podemos generar "huellas" únicas, con las cuales identificar dispositivos aprovechando las diferencias en las propiedades físicas que existen entre ellos.

Después de lo expuesto, se puede concluir que la motivación para realizar este trabajo es el estudio del concepto de funciones físicas inclonables, profundizando en su aplicación a FPGAs, además del aprendizaje de las herramientas necesarias para su diseño.

1.2 Objetivos

En este trabajo se implementará y analizará la respuesta de una función física inclonable (PUF) en FPGA. Más concretamente, se ha seleccionado una PUF basada en Ring Oscillator (RO).

Para realizar el análisis se realizarán diferentes pruebas con la PUF seleccionada en varias FPGAs para obtener diferentes datos. Con los datos obtenidos se realizarán varias comprobaciones.

- Se comprobará el efecto de la lógica del circuito en la frecuencia de oscilación de los RO, se determinará si hay una gran influencia de la lógica del circuito en la frecuencia de los ROs y si es el caso agruparla en una zona lejana de la FPGA para minimizar su influencia.
- Se comprobará como influye el posicionamiento, en la FPGA, de los ROs en su frecuencia de oscilación y se determinará en que zona de la FPGA se posicionarán para obtener una frecuencia de oscilación de los ROs, más estable.
- Se comprobará como influye la actividad electrónica generada por los ROs, en su frecuencia de oscilación para determinar cómo se verá influida la respuesta de la PUF.
- Se realizará una comparación del método clásico para la obtención de la salida de la PUF, basado en unos y ceros, frente a un método propuesto basado en calcular la diferencia entre frecuencias de ROs.
- Se comprobará, si es posible diferenciar las FPGAs utilizadas para la obtención de datos, mediante las salidas generadas por el PUF elegido.

1.3 Estructura del documento

Este trabajo se separa en diferentes capítulos cuyo contenido será explicado en este apartado.

- **Capítulo 1**: en este capítulo se introduce al lector de forma generalizad el concepto de PUF, la motivación para realizar este trabajo y los objetivos marcados para el mismo.
- **Capítulo 2**: en este capítulo se explica de forma detallada el concepto de PUF, sus propiedades, algunos tipos de PUF y sus aplicaciones. También, se explicar el tipo de PUF en el que se basa este trabajo.
- **Capítulo 3**: en este capítulo se cita la normativa aplicable a este trabajo.
- **Capítulo 4**: en este capítulo se detalla todo el proceso de creación del ROPUF, las herramientas necesarias para su implementación y las pruebas realizadas.
- **Capítulo 5**: en este capítulo se explica cómo se ha tratados los datos obtenidos y se analizan los resultados experimentales obtenidos.
- **Capítulo 6**: en este capítulo se presentan las conclusiones alcanzadas después de analizar los resultados experimentales y se presenta las líneas futuras de este trabajo.
- **Capítulo 7**: en este capítulo se detalla la planificación seguida para realizar el trabajo y se ha calculado el presupuesto para realizar el trabajo.

2 Estado Del Arte

En este capítulo se presenta el estado de arte de los PUFs, poniendo el foco en sus propiedades, aplicaciones y los diferentes tipos de PUFs.

2.1 Funciones Físicas Inclonables (PUFs)

En la actualidad, se pueden encontrar múltiples documentos tratando el tema de las PUFs, por lo que hay múltiples definiciones. Pero si se quisiera definir el concepto PUF en una sola frase seria: "Una PUF es la huella dactilar de un objeto". Las PUFs son similares a las huellas dactilares por más de una razón:

- La huella dactilar de una persona es una característica que expresa individualismo. En un sentido más inanimado, una PUF significa lo mismo para ciertos tipos de objetos. Por lo tanto una PUF es una característica identificativa de ciertos tipos de objetos.
- Como una característica individual, una huella dactilar también es inherente. Todos los seres humanos nacen con huellas dactilares, a diferencia de otras cualidades identificativas como el nombre o la firma que se adquieren después del nacimiento. Del mismo modo, las PUFs son inherentes y están presentes en un objeto desde su creación, como resultado de variaciones únicas durante el proceso de creación.
- Finalmente, como una característica individual inherente, las huellas dactilares también son inclonables. El proceso físico y biológico que determina las huellas dactilares de un ser humano está más allá de cualquier control que permita crear un segundo individuo con las mismas huellas dactilares. Esto también se aplica a seres humanos que compartan el mismo material genético, como pueden ser dos gemelos idénticos. En este sentido las PUFs son muy similares a las huellas dactilares, ya que como su propio nombre indica (funciones físicas inclonables) la inclonabilidad es una de sus principales propiedades [1].

Después de lo expuesto, una posible definición para una PUF es: "una PUF es la expresión de una característica individual, inherente e inclonable de un objeto físico".

La primera descripción del concepto PUF se puede encontrar en la tesis de Pappu escrita en 2001 [2]. Pappu utilizaba el término POWF (Physical One-Way Function) y lo definía como función fácil de computar pero difícil de invertir. El sistema físico era difícil de clonar y simular la interacción física es demasiado exigente. Más adelante Gassend et al [13] proponía el concepto PRF (Physical Random Function), pero para evitar confusión con el término Pseudo Random Function (también abreviado PRF), se utilizó el término PUF.

En la actualidad el término PUF está muy extendido y se denomina así a varias construcciones y conceptos que comparten una serie de propiedades. Algunos de estos conceptos fueron propuestos antes de que el término PUF fuese utilizado y por lo tanto no se denominaban así desde el principio.

Como el término PUF indica, PUF es una función inclonable. Es inclonable debido a variaciones aleatorias e incontrolables que surgen durante el proceso de fabricación, esto provoca que cada dispositivo tenga unas propiedades físicas únicas.

También al ser una función debe tener algunas características de funciones, por lo que dada una entrada se debería obtener una salida correspondiente. Pero no es estrictamente una función matemática, porque una PUF puede producir múltiples salidas para una entrada o incluso producir una salida para varias entradas. Este comportamiento se debe a variaciones aleatorias que pueden surgir por variaciones en las condiciones físicas.

En resumen, PUF es una función que nos da para cada entrada una salida correspondiente. Esta salida puede variar en el tiempo debido a las condiciones de operación del dispositivo y al propio envejecimiento del dispositivo. Sin embargo, las salidas deben ser lo suficientemente similares para que podamos reconocer que la salida obtenida corresponde a la entrada dada. Así mismo, necesitamos que las salidas de la PUF sean únicas para cada dispositivo. Por lo que para una entrada debe haber una salida diferente para cada dispositivo. La diferencia entre las salidas de los dispositivos debe ser lo suficientemente grande para que basándose en ellas se pueda identificar o autenticar los dispositivos [1].

Ambos requerimientos (similitud de las salidas en un mismo dispositivo y una gran diferencia en las salidas de varios dispositivos) implican que la salida de la PUF debe ser estable y única.

2.2 Propiedades de PUFs

Realizable

Una condición necesaria para la PUF y sus propiedades es que sean realizables. Se requiere que el propósito de la PUF sea al menos realizable dentro de las leyes físicas. Aunque desde un punto de vista más práctico, está más relacionado con el coste de producir la PUF. También hay una gran diferencia si se requiere que la PUF tenga un comportamiento entrada-salida determinado o no. Si es una PUF aleatoria, sin ningún comportamiento entrada-salida determinado, es fácil construir la PUF. Sin embargo, si se desea que La PUF tenga un comportamiento deseado puede llegar ser muy difícil de implementar [12].

Evaluable

Se considera que una PUF es evaluable, si para una entrada aleatoria es fácil evaluar la salida correspondiente. Es decir que se pueda evaluar en términos de tiempo, energía, potencia, área y coste [12].

Reproducible

Para una PUF y una entrada dadas en un dispositivo. Se debería obtener con una alta probabilidad la misma salida, cuando la entrada se repite varias veces. Las salidas se ven afectadas por las condiciones físicas por lo que se puede producir algún error en la salida obtenida cuando la entrada se repita durante mucho tiempo. Por este motivo, si para una entrada repetida muchas veces existe un número de errores lo suficientemente bajo en las salidas, se considera que esta la salida es la “misma” en todos los casos [12].

Única

Para una PUF y una entrada dadas, se observa la salida de diferentes dispositivos. La salidas de los diferentes dispositivos deben ser lo suficientemente grandes, con una alta probabilidad [12].

Físicamente inclonable

Dado que una PUF se basa en variaciones aleatorias e incontrolables que surgen durante el proceso de fabricación, es imposible fabricar dos dispositivos idénticos que contengan PUFs que puedan mostrar un comportamiento entrada-salida idéntico.

Con esta propiedad, ni siquiera el fabricante, que podría manipular el proceso de fabricación, puede crear dos dispositivos idénticos debido es las influencias incontrolables que surgen en el proceso de fabricación [12].

Impredecible

Es necesario conseguir que las salidas de la PUF sean impredecibles, para asegurarse de que un atacante que ha observado cierta cantidad de entradas con sus correspondientes salidas, pueda construir un modelo con el cual predecir las salidas de una nueva entrada [12].

Matemáticamente inclonable

En el caso anterior se consideraba que el atacante tenía acceso a un número limitado de muestras entradas-salidas, para predecir las futuras salidas. Pero podría darse el caso de que el atacante consiga acceso ilimitado a la PUF, y obtener todas las muestras de entradas-salidas almacenadas en la PUF. Sí las salidas permanecen siendo impredecibles se puede considerar que la PUF es matemáticamente inclonable. Para conseguir esto, el número de las muestras de entradas-salidas deben debe ser muy grande (preferiblemente exponencial) para que no sea posible almacenarlo [12].

Inclonabiladad pura

Esta propiedad se cumple cuando una PUF es inclonable tanto físicamente como matemáticamente [12].

2.3 Aplicaciones de PUFs

2.3.1 Identificación de dispositivos

La identificación es una característica inherente de la PUF, ya que al igual que se puede identificar una persona por sus huellas dactilares, con una PUF se puede identificar cualquier dispositivo basándose en sus propiedades físicas únicas.

Como ya se ha explicado anteriormente se puede producir errores en la salida debido a las condiciones físicas, es decir, la salida de la PUF de un dispositivo puede no ser la misma siempre. Sin embargo, esto no afecta a la identificación ya que las salidas de la PUF en un mismo dispositivo serán muy similares y al mismo tiempo diferentes de las salidas de otros dispositivos.

Durante el proceso de identificación, la PUF genera una salida que se compara con las salidas de PUFs de otros dispositivos almacenadas en una base de datos. Si la salida se parece lo suficiente a alguna de las salidas almacenadas y al mismo tiempo es lo suficientemente diferente del resto, se puede concluir que el proceso de identificación es un éxito. Normalmente, La similitud entre dos salidas se determina mediante la distancia de Hamming. Para que la identificación sea exitosa se deben de dar dos condiciones:

1. La distancia de Hamming entre la salida de la PUF de un dispositivo y una salida almacenada en la base de datos, debe ser menor que la distancia límite fijado. Esto conlleva que las salidas son lo suficientemente similares.
2. La distancia de Haming entre la salida de la PUF del dispositivo y el resto de dispositivos almacenados en la base de datos, debe ser mayor que las distancia límite fijado. Esta conlleva que las salidas son lo suficientemente diferentes.

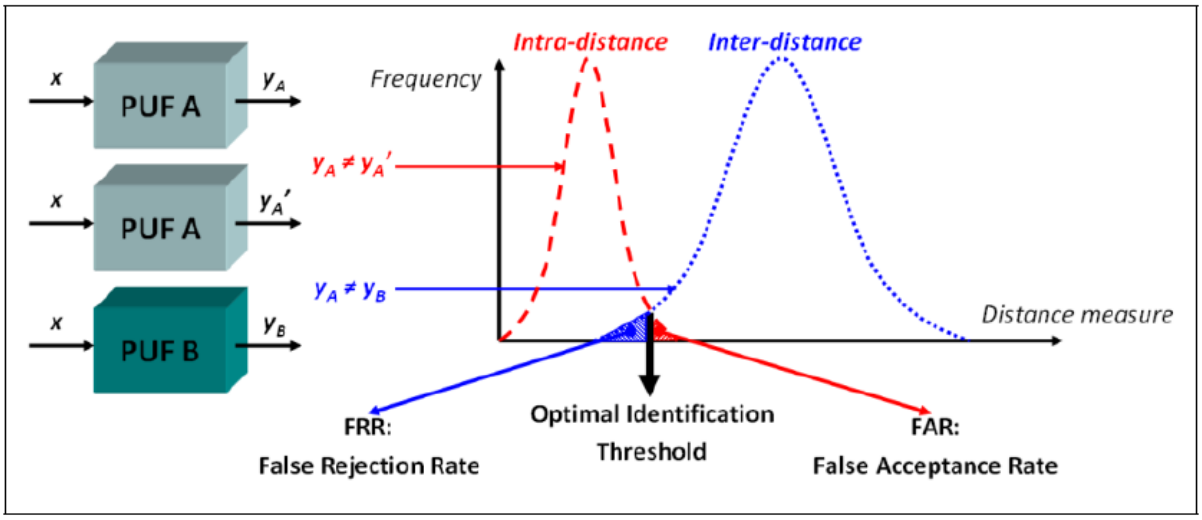

Figura 1: Principio de identificación [18]

En la Figura 1 se muestra el principio de identificación. Se muestran dos curvas una roja (intra-distance) y otra azul (inter-distance), en las que se representa la frecuencia en función de la distancia Hamming. La curva roja representa la distancia de Hamming de las salidas de PUF generadas por un mismo dispositivo, mientras que la azula representa la distancia de Hamming entre las salidas de PUF generadas por diferentes dispositivos.

Si las curvas no se solapan, se puede conseguir una identificación éxitos colocando la distancia límite entre ambas curvas. Pero si se solapan, hay fijar la distancia límite en la intersección de ambas curvas para conseguir una identificación óptima [18].

2.3.2 Autenticación

La autenticación utilizando PUFs, se realiza analizando pares de entrada-salida y toma ventaja de la unicidad e impredecibilidad de las salidas de la PUF.

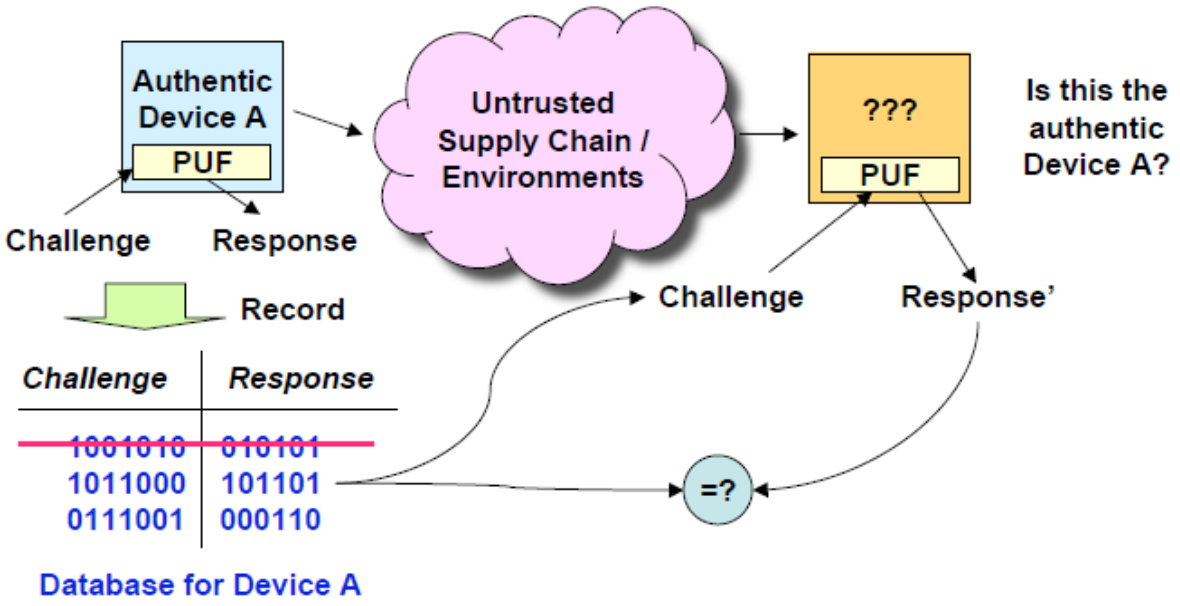

Figura 2: Esquema de autenticación con PUF [14]

En la Figura 2 se muestra un posible esquema de autenticación que consta de dos pasos [14]:

1. Se asigna un número de identificación a cada dispositivo, y después se toman suficientes muestras de entrada-salida de la PUF. El número de identificación del dispositivo junto con sus respectivas muestras de entrada-salida se almacenan en una base de datos.

2. Cuando se recibe un dispositivo para ser autenticado, se comprueba que su número de identificación se encuentra en la base de datos. Una vez hallado el número de identificación, se selecciona una de sus correspondientes muestras entrada-salida y se envía la entrada al dispositivo, si la salida generada por la PUF del dispositivo se parece suficientemente a la de la muestra almacenada, el dispositivo es autenticado de forma exitosa. Una vez finalizada la autenticación, se elimina de la base de datos la muestra utilizada en el proceso.

2.3.3 Generación de claves criptográficas

Actualmente son muchas las aplicaciones de seguridad que utilizan claves criptográficas. Normalmente estas claves se almacenan en memorias no volátiles (NVM), sin embargo para proteger la clave de forma segura ante posibles ataques se requieren soluciones que pueden llegar a ser muy complejas y caras.

Utilizando PUFs se puede conseguir una solución alternativa más barata y eficiente. En lugar de almacenar las claves en una memoria, se generan con una PUF cuando se requieran. Sin embargo, como ya se ha mencionado anteriormente, las salidas de la PUF pueden dar a

error y pueden no ser siempre las mismas cuando se generan repetidamente, debido a las variaciones en las condiciones físicas o ruidos aleatorios. Por lo tanto, las salidas de la PUF deben ser estables antes de ser utilizadas como claves. Esto se consigue mediante códigos de corrección de errores (ECC), que corrigen los bits erróneos de la salida. Por lo tanto, combinando PUFs y ECC se pueden generar claves aleatorias, impredecibles y estables.

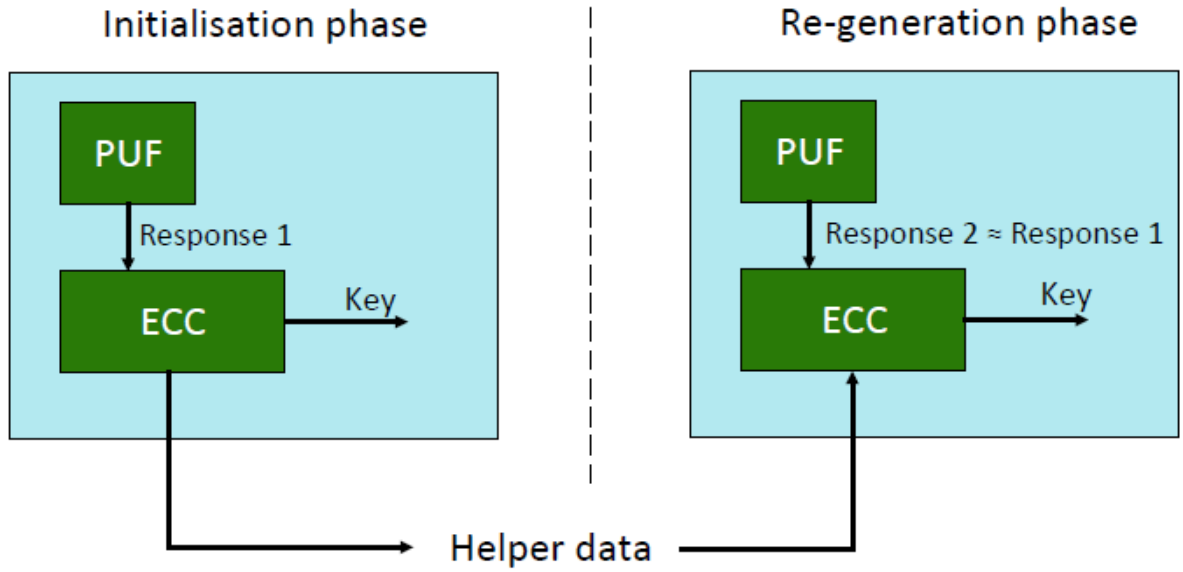

Figura 3: Esquema del proceso de generación de claves criptográficas [14]

En la Figura 3 se puede observar el esquema de un posible proceso de generación de claves criptográficas. Este proceso de divide en dos fases:

1. En la primera fase, denominada fase de inicialización, una PUF genera la clave y el ECC produce datos auxiliares que se utilizarán posteriormente para corregir la salida de la PUF. Los datos auxiliares son públicos, así que no debería ser posible descifrar la clave a partir de estos.
2. En la segunda fase, denominada fase de regeneración, se vuelve a generar la clave cuando es requerida. La PUF genera una salida, después la salida se procesa por el ECC y se corrige utilizando los datos auxiliares obtenidos en la fase anterior. Después de la corrección se obtiene la misma clave que en la fase anterior [14].

2.4 Tipos de PUFs

En este capítulo se describirán algunos tipos de PUFs, no se presentarán todos los tipos de PUFs debido a que existe una gran cantidad.

2.4.1 Optical PUF

Las Optical PUFs fueron propuestas en 2002 por Pappu con el nombre de physical one-way functions (POWF) [2].

Las optical PUFs, constan de un medio óptico transparente (Optical token) llenado por una gran cantidad de partículas de dispersión de luz. Cuando un rayo láser ilumina al medio óptico, surge una mancha con un patrón aleatorio y único. La aleatoriedad en este tipo de PUF surge debido a la colocación aleatoria de las partículas de dispersión de la luz en el medio óptico en el proceso de fabricación. La mancha resultante se graba y codifica en una cadena de bits que representa la salida de la PUF. En este tipo de PUF la entrada es el ángulo en el que se fija el rayo láser que impacta al medio óptico [18].

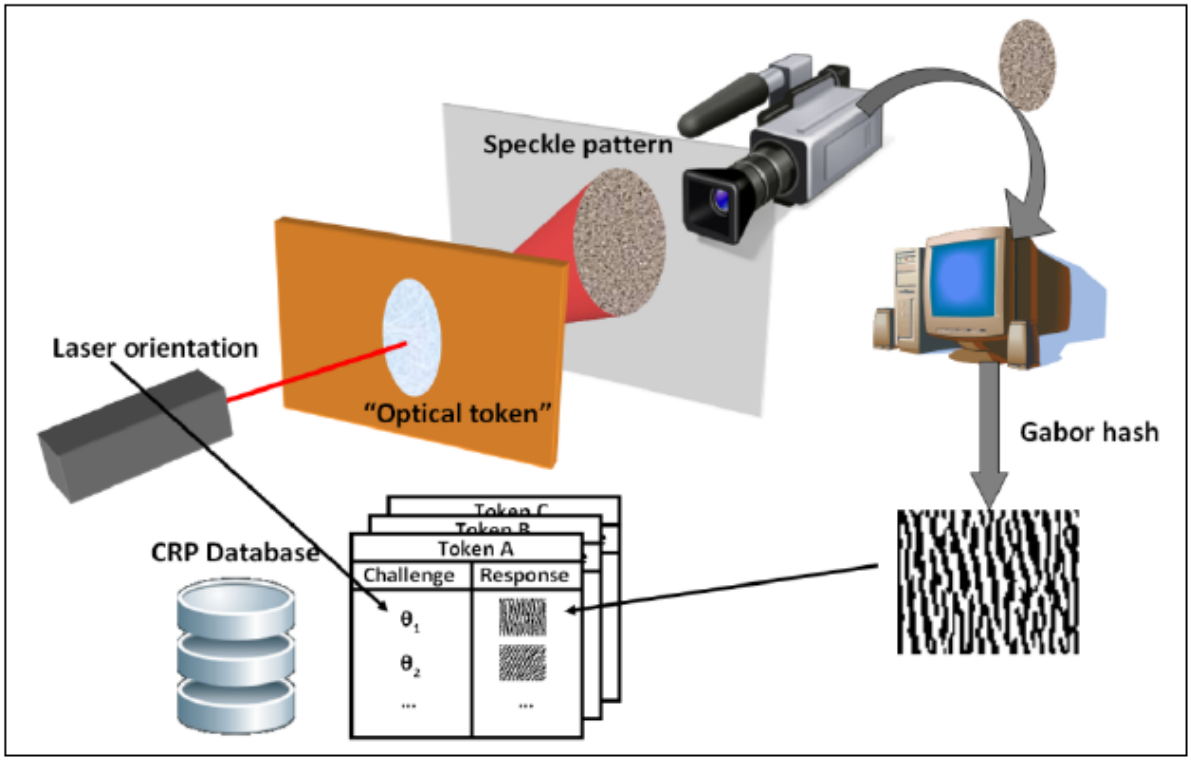

Figura 4: Esquema de funcionamiento de las Optical PUFs [18]

En la Figura 4 se puede observar de forma gráfica el funcionamiento de una Optical PUF explicado en el párrafo anterior.

2.4.2 Coating PUF

El concepto de Coating PUF es introducido por Tuyls et al. En [6], consiste en cubrir un circuito integrado con un revestimiento protector. El material de revestimiento es llenado con partículas dieléctricas de tamaño, forma y localización aleatorias. Debajo del revestimiento se colocan sensores de alambres de metal para medir la capacitancia del revestimiento. Al medir la Coating PUF desde el exterior se obtendrá valores de capacitancia diferentes ya que las medidas son muy sensibles a la localización de las partículas dieléctricas.

Las Coating PUF ofrecen una gran protección ante ataques físicos, porque la manipulación del revestimiento provoca cambios en la capacitancia, lo que cambia totalmente la salida de la Coating PUF.

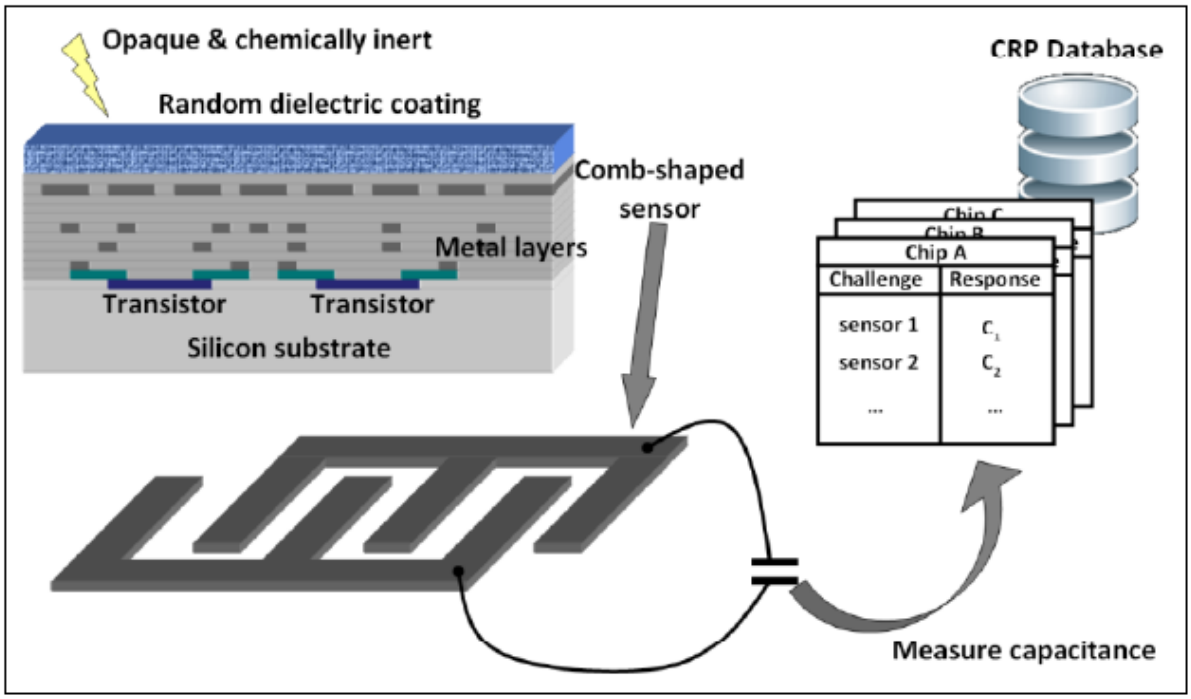

Figura 5: Esquema de funcionamiento de las Coating PUFs [18]

En la Figura 5 se puede observar de forma gráfica el funcionamiento de una Coating PUF, como se ha explicado en el párrafo anterior.

2.4.3 SRAM PUF

La SRAM es una memoria estática basada en biestables y flip-flops que se utilizan para almacenar la memoria.

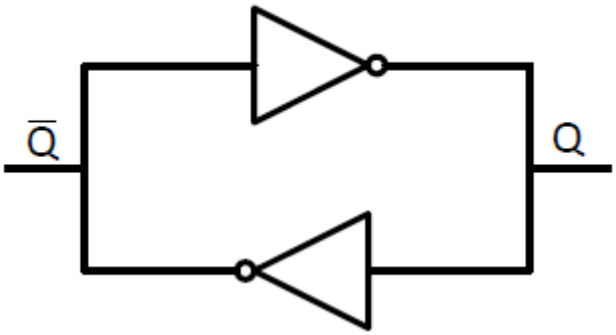

Figura 6: celda de SRAM [1]

En la Figura 6 se puede observar una celada de SRAM. Consta de un lazo cerrado formado por dos inversores. El circuito tiene dos estados posibles, 0 y 1, que representan el valor binario almacenado en la memoria [1].

Las SRAM PUFs se basan en el contenido de la memoria después de encenderse, es decir el estado de cada celda cuando se enciende. Al encenderse la memoria, algunas celdas tienen preferencia a tomar el estado 1, otras 0 y alguna no tienen preferencia a tomar ningún estado. Esta preferencia no puede ser controlada o influenciada y es totalmente aleatoria, debido a desajuste físicos en la celda que surgen durante el proceso de fabricación. Las celadas con preferencia a tomar el estado 1 ó 0 se consideran celdas estables, mientras que las que no tienen preferencia se consideran inestables.

Con las celdas estables se pueden identificar diferentes dispositivos cuando se utilizan como salida de la SRAM PUF, mientras que las celdas inestables causan errores en la salida de la SRAM PUF. Esto se puede observar de forma gráfica en la Figura 7, donde se muestra en una escala de 0 a 1, las probabilidades de que una celda sea 1 [8].

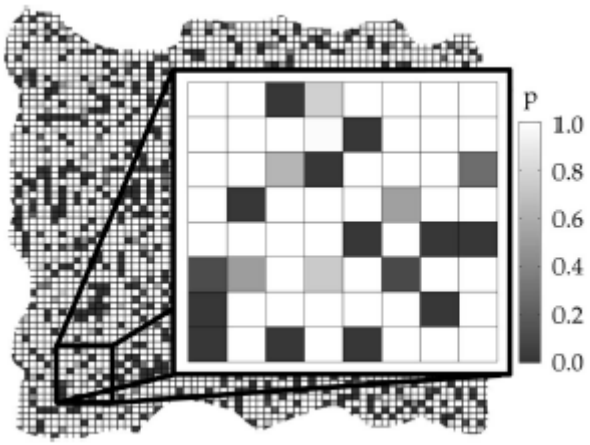

Figura 7: mapa de celdas de una SRAM [7]

2.4.4 Butterfly PUF

En la mayoría de los casos es imposible implementar las SRAM PUF en FPGAs, debido a que la SRAM se inicializa con valores predefinidos y por tanto se pierde la aleatoriedad. Otra desventaja, es a la hora de generar la salida de la PUF la SRAM debe ser leída justo después de encenderse y antes de que se sobrescriba. Estos problemas fueron los principales motivos del surgimiento de la Butterfly PUF, que se basa en un circuito con lazo cerrado de dos inversores [9].

El concepto de la Butterfly PUF consiste en simular el comportamiento de la SRAM PUF después de encenderse y estabilizarse en un estado. Para ello se imita el circuito de una celda de SRAM, que consta de un lazo cerrado de dos biestables, en los que se conecta la salida de uno con la entra del otro, para conseguir así simular su funcionamiento.

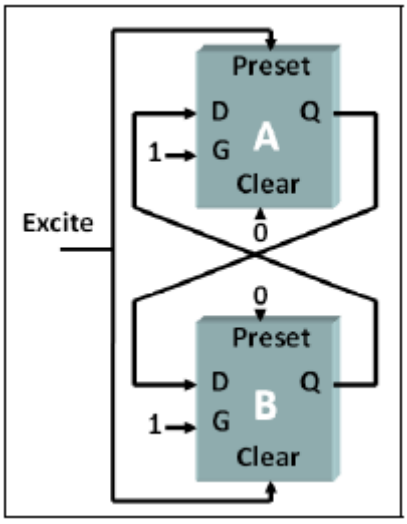

Figura 8: circuito simétrico de una Butterfly PUF [18]

En la Figura 8 se puede observar el circuito de una Butterfly PUF, y como se ha comentado, al igual que una celda de SRAM tomo dos estados, 0 ó 1. Sin embargo, utilizando las señales Clear y Preset se provocar un estado inestable para después volver a un estado estable. A igual que ocurre con cada celda de la SRAM que tiene una preferencia a tomar un estado, una celda de la Butterfly PUF también tiene esta preferencia que se determina por a diferencias físicas entre los biestables y lazo que los une [9].

2.4.5 Arbiter PUF

La Arbiter PUF se propone inicialmente en [10] y [11]. La idea es realizar una carrera digital utilizando dos caminos de un circuito y tener un "arbitro" que decida qué camino es más rápido. Si los dos caminos se diseñan de manera simétrica, con los mismos retardos, se asegura un resultado aleatorio. Durante el proceso de fabricación, se producen variaciones que afectan a las propiedades físicas que determinan el retardo exacto de cada camino, y

por tanto aunque los caminos teóricamente son exactos debido a estas variaciones existe una diferencia entre los retardos de los dos caminos. Dependiendo del resultado de la carrera el árbitro genera un bit de salida para la PUF. El árbitro consiste en un circuito lógico que determina el camino más rápido.

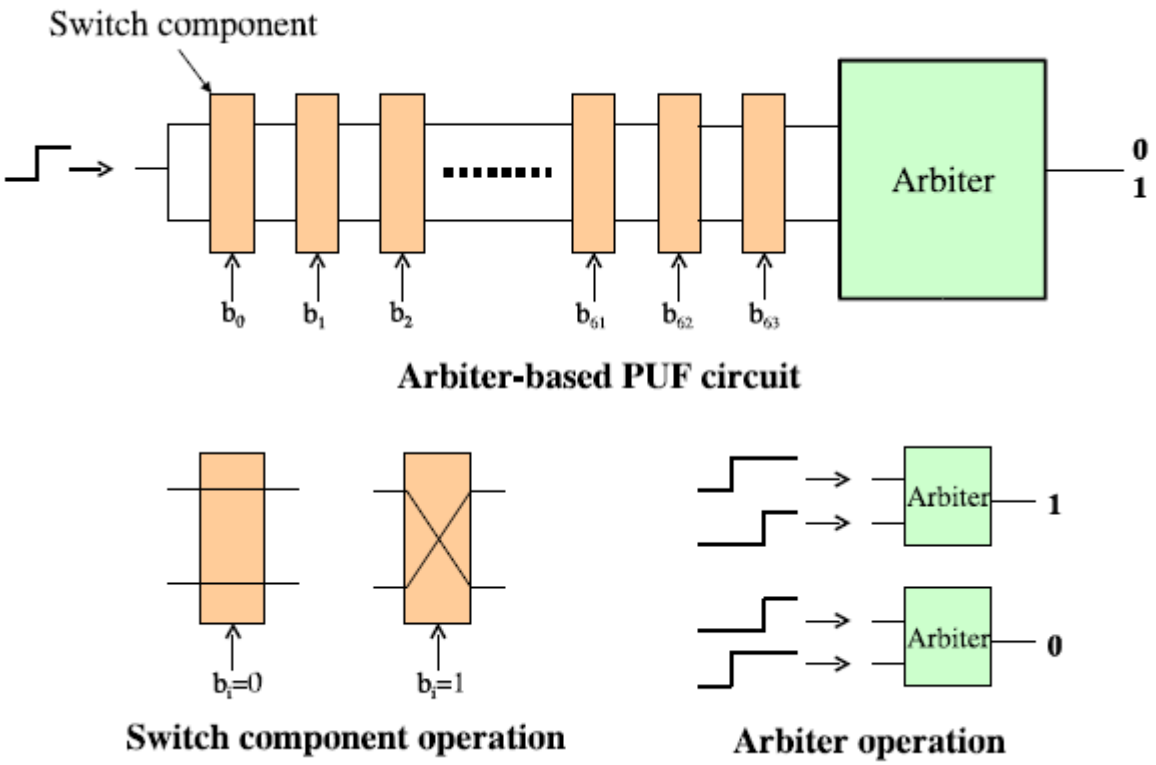

Figura 9: Esquema de la Arbiter PUF [10]

En la Figura 9 se puede observar el esquema de una Arbiter PUF. Los caminos se conforman por bloques de conmutadores (switch) en serie, en los cuales se conecta las dos señales de entrada con las salidas en diferente configuración según el bit de control. Si el bit de control es b=0, la entrada se conecta directamente con la salida correspondiente de forma lineal, y si es b=1 las entradas se conectan con la salidas de forma cruzada. Los conmutadores se pueden realizar con multiplexores. Al final del recorrido, hay un árbitro que detecta que señal llega antes, esto se realiza con un biestable.

Si se dispone de n bloques de conmutadores colocadas en serie para formar los caminos, se obtendrán 2^n posibles configuraciones. El resultado de cada configuración es un único bit si se desea obtener más bits, hay dos formas posibles de hacer y que se pueden combinar. La primera, es realizar más circuitos y con la misma configuración cuando se obtiene la salida de la PUF. La segunda, es realizar varias "carreras" con el mismo circuito y ordenar los bits de salida para generar la salida de la Arbiter PUF.

2.4.6 Ring Oscillator PUF (ROPUF)

En este apartado se describirá brevemente el concepto PUFs basadas en Ring Oscillator (RO), pero más adelante se explicará más en detalle ya que en este trabajo de fin de grado la PUF utilizada se basa en RO.

La ROPUF es una PUF basada en retardos, explota las variaciones aleatorias en los retardos de puertas lógicas y sus interconexiones. Para medir los retardos, se construye un circuito con puertas lógicas y se realimenta la salida para conseguir hacer que oscile.

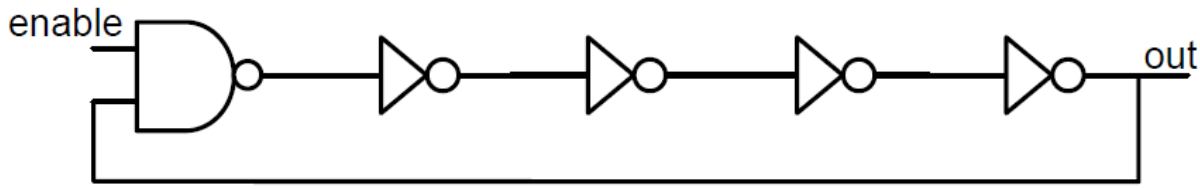

Figura 10: Circuito Ring Oscillator [12]

En la Figura 10 se puede observar un circuito básico de RO, consiste en una puerta NAND y cuatro puertas not para invertir la salida y conseguir que oscile al realimentarla.

Utilizando contadores podemos medir las oscilaciones del RO en un periodo de tiempo, y a partir de esa medida se puede calcular la frecuencia del RO. Las frecuencias medidas reflejan las variaciones aleatorias en los retardos del circuito. Por lo tanto, estas frecuencias se pueden utilizar para crear la salida de una PUF [13].

2.5 PUF basada en Ring Oscillator

La PUF que se utiliza en este trabajo de fin de grado se basa en Ring Oscillator (RO), como ya se ha explicado anteriormente, un RO es un circuito de puertas lógicas realimentado oscilante, donde se cuentan las oscilaciones en un periodo de tiempo para calcular la frecuencia que se utiliza para crear la salida de la PUF.

El primer tipo de PUF se propone por Gassend et al. En [13]. Las frecuencias medidas de ROs iguales en diferentes dispositivos muestran suficiente diferencia para ser consideradas como una salida de PUF. Sin embargo, las condiciones del ambiente tienen un influencia sobre la frecuencia de la PUF que debe ser compensada. Para compensar esta influencia Gassend et al. propuso una técnica llamada medición compensada (“compensated measuring”), esta técnica se sostiene que la variaciones en el ambiente afectan de manera similar la frecuencias de Ros, por lo tanto se puede utilizar la relación entre dos pares de ROs como la salida de la PUF.

Aunque con esta técnica se consigue compensar de manera efectiva la influencia de las condiciones ambientales, la ROPUF tiene otro inconveniente. Al igual que ocurre con la Arbiter PUF, puede ser vulnerable a ataques para crear un modelo matemático que simule

el funcionamiento de la PUF, y por tanto deben tomarse medidas para protegerse contra estos ataques.

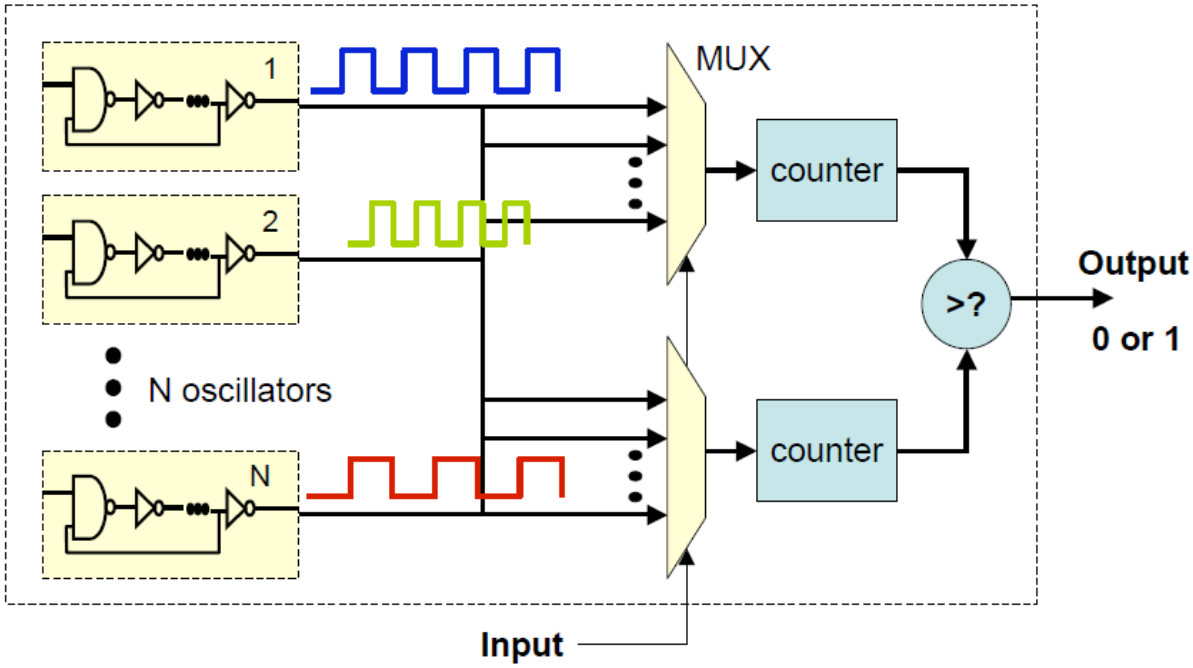

Figura 11: Diseño de Ring Oscillator PUF [14]

En la Figura 11 se puede observar otra construcción de ROPUF propuesta por Suh y Devadas [14]. Consiste en n ROs simétricos conectados a dos multiplexores, cada multiplexor selecciona un RO según su entrada (Input) y se conecta su salida a un contador. Ambos contadores cuentan las oscilaciones de los ROs para un intervalo de tiempo fijo. Los resultados de los contadores se comparan y se genera un bit de la salida de la PUF según el resultado de la comparación. Este proceso debe repetirse muchas veces con diferentes pares de RO para obtener una salida de la PUF completa, es decir, una salida con el número de bits igual al número de comparaciones realizadas.

Es necesario que todos los ROs en la ROPUF sean simétricos entre ellos para asegurar que los resultados de las comparaciones sean impredecibles. Si los ROs son simétricos aseguramos que las diferencias en las frecuencias son completamente dependientes de variaciones aleatorias en los retardos, que surgen debido a variaciones en el proceso de fabricación. La comparación entre frecuencias también puede considerarse otra forma de medición compensada para eliminar la influencia de las condiciones ambientales.

Suh y Devadas también propusieron una técnica llamada “1-out-of-k masking”. Esta técnica reduce el número de posibles comparaciones para obtener una salida más estable y consiste en seleccionar un par de RO con la mayor deferencia entre sus frecuencias.

Una de las desventajas de esta construcción de ROPUF, es que el número de posibles comparaciones es limitado si se busca que los bits de la salida de la PUF sean independientes. El número máximo de comparaciones con n ROs que podemos realizar con este método es $\binom{n}{2} = \frac{n(n-1)}{2}$. Sin embargo la entropía es menor que $\binom{n}{2}$, porque existe una correlación entre los bits obtenidos. Por ejemplo, si RO A es más rápido que RO B y RO B es más rápido que RO C, se puede deducir que RO A es más rápido que RO C [14].

Para determinar la máxima entropía de este diseño, es decir, el máximo número de bits independientes generados por comparaciones de pares, hay que considerar todas las ordenaciones posibles de n ROs. Existen $n!$ posibles ordenaciones de ROs y si todas las ordenaciones tienen la misma probabilidad, la entropía será de $\log_2(n!)$.

Una manera fácil de obtener bits independientes en la salida de la PUF es utilizar cada RO solo una veza para cada comparación, así el número de bits obtenidos a la salida será $\frac{n}{2}$. Combinado con la técnica de "1-out-of-k masking", el número de de bits de la salida se reducirá significativamente.

Otra técnica para elevar la estabilidad de la salida en ROPUF fue propuesta por Yin and Qu [15]. Los ROs se dividen en dos grupos mutuamente excluyentes, es decir, un RO no puede estar en ambos grupos, y la comparación de frecuencias se realiza solo entre ROs del mismo grupo, donde se garantiza una alta estabilidad en los resultados de las comparaciones. La división de ROs en grupos se realiza con el algoritmo LISA (Longest Increasing Subsequence-Based Grouping Algoritm). Para cada RO, se realizan mediciones en diferentes condiciones y se almacena los valores máximos y mínimos de la frecuencia. Con el algoritmo LISA, los ROs se dividen en grupos según si la diferencia de los valores mínimos y máximos entre pares de RO sobrepasen un límite fijo o no.

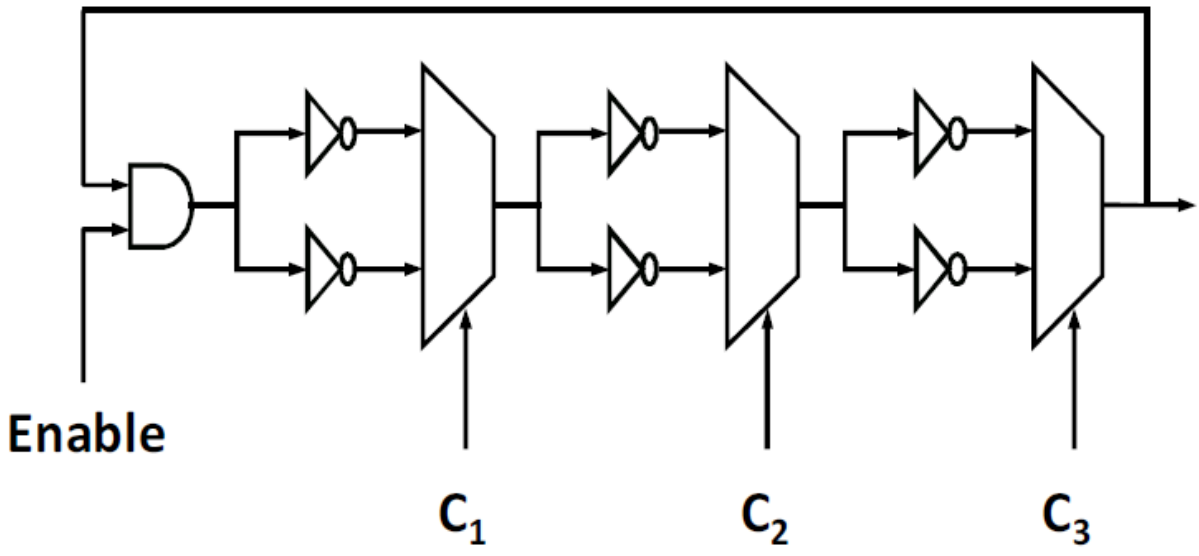

Figura 12: Configurable Ring Oscillator [16]

En la Figura 12 se muestra el diseño de un "Configurable Ring Oscillator". Este diseño es introducido por Maiti and Schaumont en [16]. Se utiliza el mismo diseño que Suh and Devadas junto con la técnica de "1-out-of-k masking" para aumentar la estabilidad de la salida, pero el diseño está basado en ROs configurables en vez de los ROs básicos. Se considera la configuración más estable de k configuraciones de un par de ROs, no el par más estable. Esto da como ventaja una utilización de los recursos más eficiente.

El RO básico compuesto por cinco puertas lógicas, una NAND y cuatro inversores, implementado en una FPGA ocupa casi todo el CLB (Configurable Logic Block) de Spartan-3E de Xilinx. Mientras el RO configurable como el mostrado en la Figura 12, compuesto por un AND, seis inversores y tres multiplexores, ocupa un único CLB. En este caso hay tres bits de control, lo que equivale a ocho posibles configuraciones del RO. Al igual que en los otros diseños es necesario que los RO configurables sean simétricos.

En la Figura 13 se muestra otra propuesta de RO configurable en la Spartan-3E de xilinx, presentada en el trabajo de Xin et al. [17]. Es una extensión de la RO configurable de Maiti et al. [16]. La RO configurable todavía cabe en un solo CLB en una FPGA, pero ahora se pueden realizar 256 configuraciones diferentes. En este diseño se utilizan más multiplexores y también se utilizan biestables. Los biestables se utilizan como una unidad de retardo adicional. Al igual que los otros diseños los ROs tienen que ser simétricos entre ellos y utilizar los ROs a comparar en la misma configuración para que el resultado dependa de las variaciones aleatorias en los retardos.

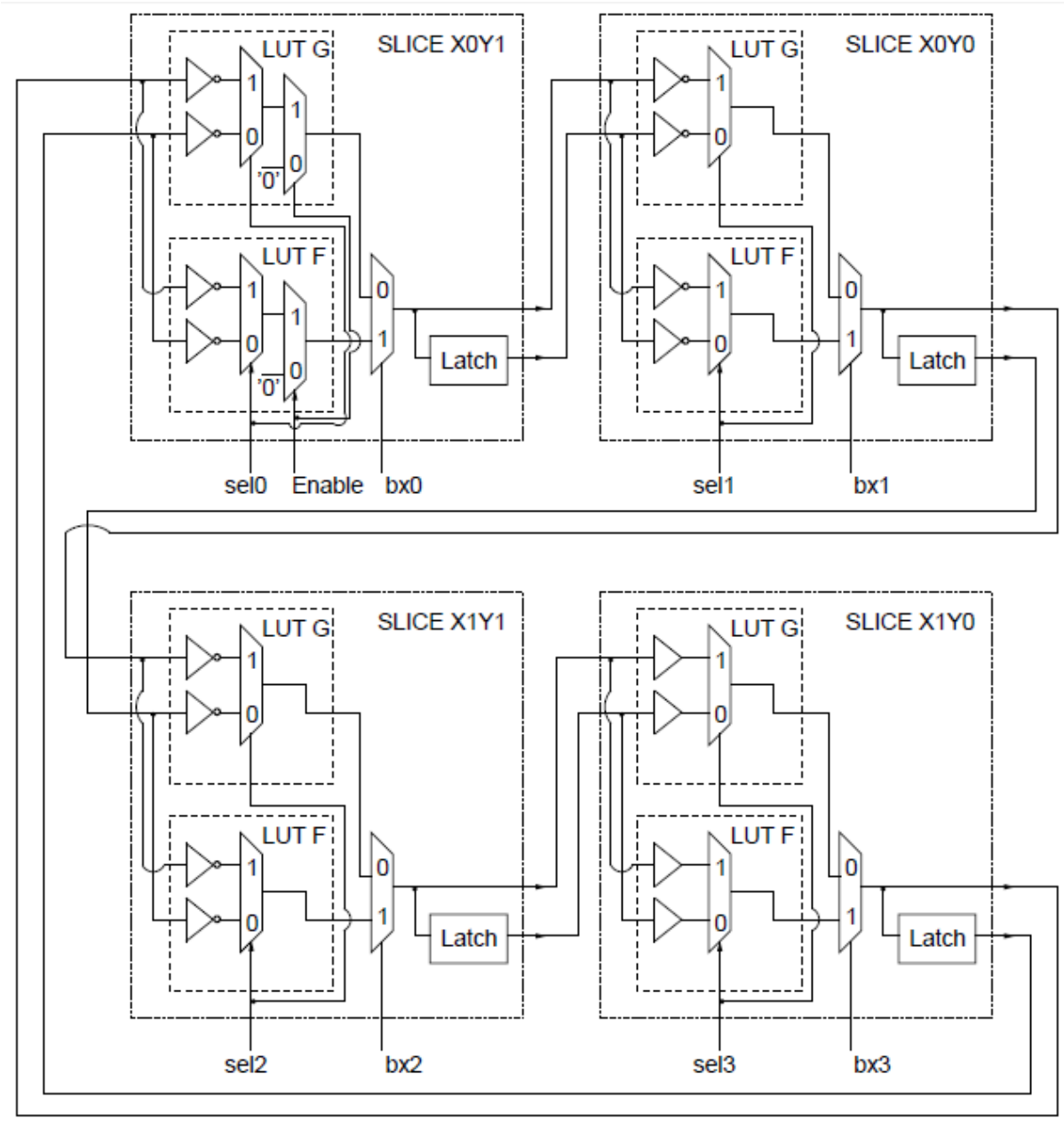

Figura 13: Configurable Ring Oscillator de 256 configuraciones diferentes [17]

3 Marco Regulador

El concepto de las funciones físicas inclonables es reciente y todavía no hay estándares definidos. Actualmente existe un estándar, el ISO/IEC 20897, en desarrollo en el que se va a especificar los requerimientos de seguridad, métodos de evaluación y tests para generar parámetros de seguridad con PUFs.

Los PUFs se utilizan en métodos de autenticación por lo que un posible estándar aplicable a los PUF sería el ISO/IEC 9798. En este estándar se especifican varios protocolos de autenticación para sistemas de seguridad.

El ISO/IEC 9798 consta de seis partes.

- **ISO/IEC 9798-1**: en esta parte se explica de forma general un modelo de autenticación propuesto para sistemas de seguridad basados en mecanismos de autenticación.
- **ISO/IEC 9798-2**: en esta parte se especifica mecanismos de autenticación basados en algoritmos de cifrado simétricos. Se especifican mecanismos de autenticación mutua entre dos entidades y mecanismos que utilizan a un tercero de confianza para establecer un secreto y poder realizar la autenticación.
- **ISO/IEC 9798-3:** en esta parte se especifican mecanismos de autenticación basados en firmas digitales.
- **ISO/IEC 9798-4:** en esta parte se especifican mecanismos de autenticación basados en funciones de verificación criptográfica.
- **ISO/IEC 9798-5:** en esta parte se especifican mecanismos de autenticación basadas en técnicas de cero pruebas.
- **ISO/IEC 9798-6:** en esta parte se especifican mecanismos de autenticación basados en transferencia de datos manual entre los dispositivos a identificar.

4 Marco Experimental

En este capítulo se va a explicar todo el proceso llevado a cabo para crear la PUF basada en Ring Oscillator y se explicarán todas las herramientas utilizadas y códigos creados para alcanzar el objetivo del trabajo.

4.1 Arquitectura

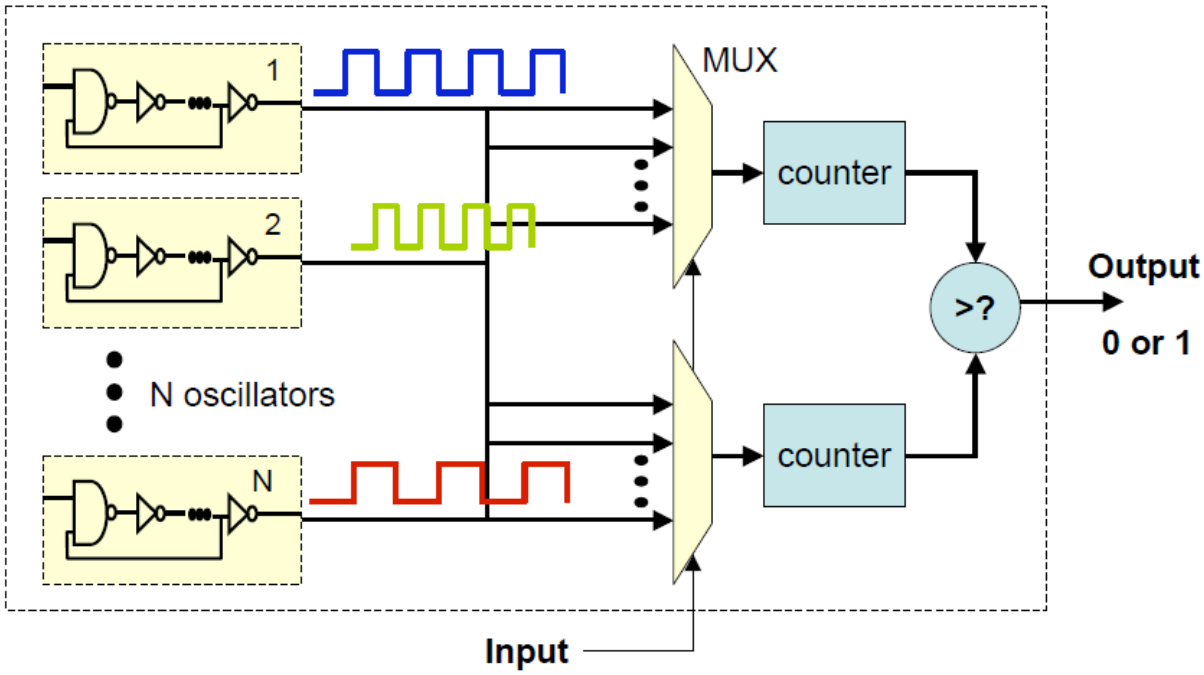

Figura 14: ROPUF implementado [14]

En la Figura 14 se muestra un esquema del ROPUF que se ha implementado y a lo largo de este apartado se explicará cómo se ha conseguido realizar las diferentes partes que conforman el ROPUF.

Para la creación de la PUF basada en Ring Oscillator, se ha utilizado el programa Xilinx. Este programa permite el diseño y simulación de circuitos, tanto realizados en forma de

esquemático como en forma de lenguaje de programación. En este caso para la creación de los circuitos del ROPUF, se han realizado códigos en VHDL y utilizado otras herramientas de Xilinx para implementarlo en placas FPGA Spartan-3E.

4.1.1 Creación del RO

El RO utilizado es un RO básico, conformando por cuatro inversores y una puerta NAND que nos permite habilitar y deshabilitar el RO, como se muestra en la Figura 15.

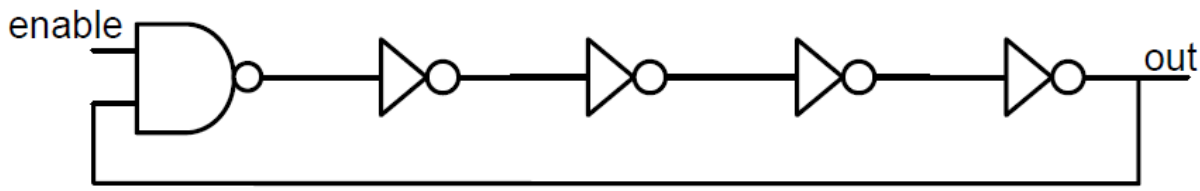

Figura 15: Ring Oscillator implementado [12]

Para realizar este circuito se ha creado un código en VHDL. En el código se ha creado una señal de entrada "Enable" y una salida ("Q"), después en un process se ha simulado el funcionamiento del circuito con un "AND" y cinco "NOT" y realimentando la salida con la entrada para conseguir que la salida oscile continuamente mientras la seña "Enable" esté activa. En el anexo 1 se muestra el código realizado.

4.1.2 Multiplexores

Como se puede observar en la Figura 14, los multiplexores, mediante los bits de control (Input), controlan que par de ROs se conectan a los contadores que cuentan las Oscilaciones. Para conseguir este funcionamiento en el código VHDL, se ha creado dos contadores que actúan como los bits de control de los multiplexores. Son contadores de seis bits, es decir cuentan hasta 64, ya que se dispone de 64 ROs.

El funcionamiento de estos contadores consiste en que cada vez que pasa un periodo de tiempo fijo, un contador aumenta en uno y una vez este haya desbordado, cuando alcanza sesenta y cuatro, entonces el otro contador aumenta en uno. De esta forma conseguimos conectar todos lo ROs a los contadores que cuentan las Oscilaciones y realizar comparaciones entre todos los ROs.

4.1.3 Contadores

Los contadores que se muestran en la Figura 14 son los encargados de contar las oscilaciones de los ROs. Para conseguir este funcionamiento se ha creado dos contadores que aumenta en uno con cada oscilación del RO al que estén conectados. Cada cierto periodo de tiempo fijo e igual al utilizado en los multiplexores, se reinician. Con esto conseguimos que durante un periodo de tiempo fijo, los multiplexores conecten dos ROs a

los contadores y que estos cuenten las oscilaciones en ese periodo y almacenar ese valor. El número de oscilaciones en el periodo de tiempo fijado, serán la frecuencia de oscilación del RO.

El periodo de tiempo fijado, en el cual se conecta los ROs y se cuentan sus oscilaciones es de $2^{18} = 262144$ ciclos de reloj del sistema. Cada ciclo de reloj del sistema es de 20ns. Por lo tanto, el periodo es de $262144 * 20\,ns = 5.243\,ms$. En los anexos 2 y 3 se muestra en detalle el código VHDL realizado.

Una vez realizado los ROs, los multiplexores y contadores solo falta hacer la comparación entre los resultados de los contadores y generar la salida del ROPUF. Esto se realizará más adelante y se analizará si es posible identificar las placas FPGA mediante esta técnica.

4.2 Spartan-3E

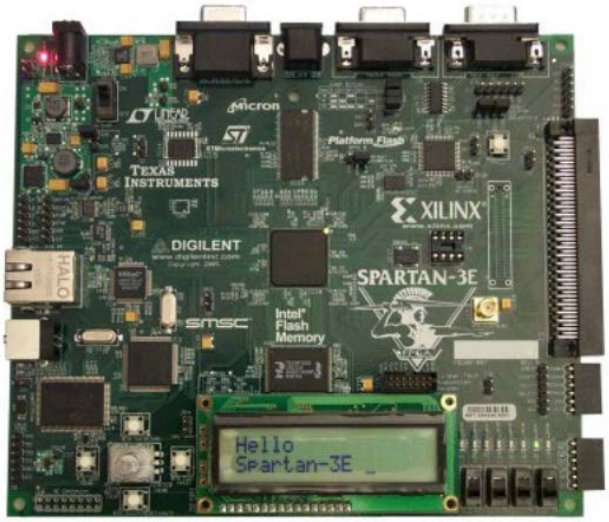

Figura 16: Spartan-3E [4]

Las pruebas del ROPUF se utilizan en placas FPGA Spartan-3E (Figura 16). Es una evolución de la Spartan-3. Se mejora el rendimiento del sistema ya que el número de pines I/O es mayor por celda lógica. Debido a la tecnología 90 nanómetros es más barata, más avanzada y permite un mayor ancho de banda.

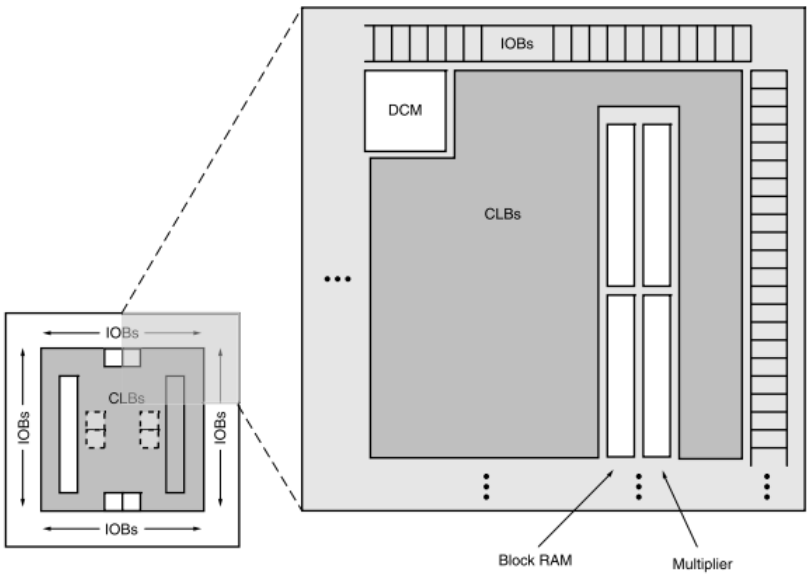

Figura 17: Arquitectura Spartan-3E [5]

En la Figura 17 se puede observar la arquitectura de la Spartan-3E, está compuesta de cinco elementos programables:

- Bloques I/O: gestiona la transmisión y recepción de datos bidireccional ente los pines I/O y el resto de componentes.
- CLBs (Configurable Logic Block): Almacenan la información en biestables y flip-flops e implementan la lógica del circuito.
- Bloques multiplicadores: calculan la multiplicación entre números binarios de 18 bits.
- Bloque de RAM: Almacenan información en bloques de 18Kb.
- Bloques DCM (Digital Clock Manager): Proporciona auto-calibración digital de las señales de reloj. Es posible distribuir, multiplicar, dividir y retrasar las señales.

4.3 Mediciones y Pruebas

Una vez Creado el ROPUF y los diferentes modos de funcionamiento se ha procedido a la realización de pruebas y mediciones en placas FPGA Spartan-3E. Se realizarán pruebas en cuatro placas Spartan-3E para obtener las frecuencias de los ROs en diferentes modos de funcionamiento y en diferentes zonas de la FPGA.

Para realizar las pruebas y almacenarlas se utilizarán diferentes herramientas de Xilinx que se explicarán a lo largo de este apartado.

4.3.1 Hard Macro

Una vez creado el RO, hay que emplazarlo en la FPGA y replicarlo en diferentes zonas de esta. Para el ROPUF es necesario que los RO sean idénticos, es decir, todos los RO emplazados en la FPGA deben ser iguales, deben tener el mismo rutado y la misma configuración de LUTs, para garantizar que las variaciones en los retardos son solo debidas a variaciones en el proceso de fabricación. Para asegurarnos de que los RO son idénticos, se ha creado una Hard Macro a partir del RO diseñado utilizando la herramienta "FPGA Editor" de Xilinx.

Una Hard Macro en un bloque de circuito reutilizable, con la Hard Macro se preserva todas las características del bloque desde el rutado, hasta la configuración de LUTs. Esto permite replicar el RO en diferentes zonas de la FPGA, asegurando que todos los RO tienen las mismas características.

Una vez creada la Hard Macro se genera un archivo .nmc que debe ser incluido en la carpeta del proyecto en el que se va a utilizar. Para ser utilizado se debe instanciar como cualquier otro componente y debe añadirse al archivo .ucf de forma manual ya que la herramienta FPGA Editor no lo añade automáticamente.

En el anexo 8 se explica en detalle los pasos seguidos para crear la Hard Macro.

4.3.2 Archivo UCF

Una vez realizado el RO, para realizar el resto de partes que forman el ROPUF se ha creado otro proyecto, y al cual se ha añadido el componente RO creado con la hard macro para poder replicarlo en diferentes zonas de la FPGA.

El número de ROs que se van a utilizar es de sesenta y cuatro. Estos ROs se distribuirán en la FPGA en forma de matriz de ocho columnas por ocho filas. Además, esta matriz se posicionara en diferentes zonas de la FPGA para evaluar cómo se ven afectadas las frecuencias de los ROs.

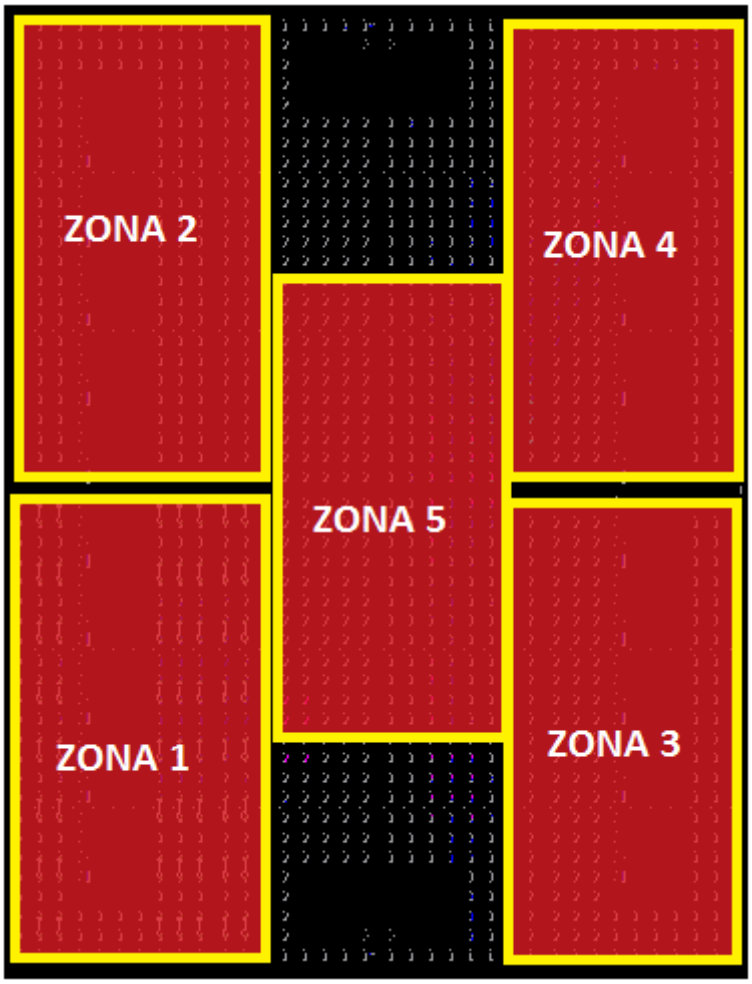

Figura 18: Zonas de la FPGA

En la Figura 18 se puede observar las diferentes zonas donde se posicionará la matriz de 8x8 ROs. Para posicionar los ROs hay que hacerlo manualmente en el fichero UCF. El archivo UCF es un archivo en el que se indican instrucciones a la FPGA sobre el mapeo, emplazamiento y tiempos.

Los RO no pueden emplazarse en cualquier posición de la FPGA, cada RO debe emplazarse en su SLICE correspondiente. Para ello, hay que añadir al fichero UCF la siguiente sentencia por cada RO de los sesenta y cuatro:

INST\“nombre del RO” Loc=SLICE_X “coordenada X del SLICE” Y “coordenada Y del SLICE”;

Donde pone nombre del RO, hay que escribir el nombre del RO que se le ha asignado en el proyecto y donde pone coordenada X Y, es donde se indica la posición del RO en la FPGA.

Para automatizar la creación del archivo UCF, se ha creado un pequeño código en C++ para crear las sentencias de los sesenta y cuatro ROs posicionados en las diferentes zonas.

Consiste en una serie de bucles for en los que se cambia las coordenadas X Y para emplazar el RO en la posición adecuada. En el anexo 4 se muestra el código C++ realizado.

Además del emplazamiento de los RO en el archivo UCF también hay que indicar el periodo del reloj del sistema así como asignar las entradas y las salidas del proyecto con las de la FPGA. En el anexo 5 se muestra el archivo UCF.

4.3.3 PlanAhead

PlanAhead es un software de diseño y análisis y que se utiliza para diseñar FPGAs. Se ha utilizado esta herramienta para crear un bloque con la lógica del circuito y emplazarla en una zona lejana a los ROs. Con esto se busca comprobar si la lógica del circuito afecta a las frecuencias de los ROs.

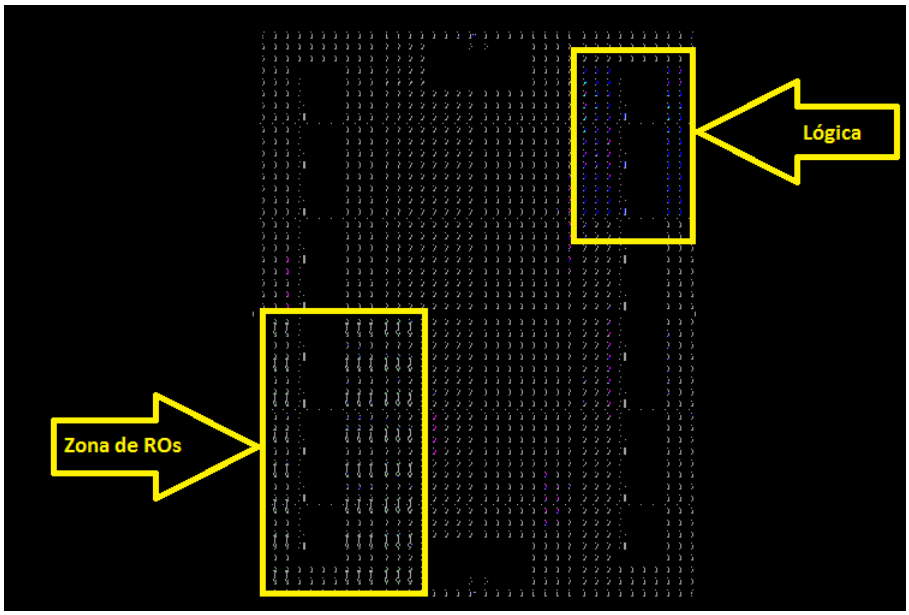

Figura 19: FPGA con lógica de circuito agrupada

En la Figura 19 se puede ver cómo queda la FPGA después de utilizar PlanAhead para crear un bloque con la lógica del circuito y emplazarlo. En la parte superior derecha (pblock_1) de la FPGA está emplazado el bloque con la lógica y en la parte inferior izquierda es donde están emplazados los ROs (zona 1). En el anexo se muestra el proceso seguido para crear el bloque y emplazarlo.

4.3.4 Modos de funcionamiento

Una vez Creado el ROPUF se ha procedido a la realización de diferentes modos de funcionamiento para comprobar cómo afecta la actividad electrónica generada por ROs a las frecuencias de los ROs bajo comparación.

7	15	23	31	39	47	55	63
6	14	22	30	38	46	54	62
5	13	21	29	37	45	53	61
4	12	20	28	36	44	52	60
3	11	19	27	35	43	51	59
2	10	18	26	34	42	50	58
1	9	17	25	33	41	49	57
0	8	16	24	32	40	48	56

Tabla 1: Matriz 8x8 ROs

Como ya se ha explicado anteriormente se ha creado de una matriz de 8x8 ROs (64 ROs en total) que se emplazará en diferentes zonas de la FPGA. De esta forma se selecciona dos de los ROs, se habilitan (con enable) para que oscilen y se conectan a los contadores para contar las oscilaciones. Para realizar las pruebas se han creado diferentes modos de funcionamiento en los que además de habilitar los dos ROs seleccionados, también se habilitan otros ROs para ver como se ve afectada la frecuencia de oscilación. Según el modo de funcionamiento se controlan determinados enable de los ROs, en el código de la creación del ROPUF, que no están conectados a los contadores para hacer que oscilen o no.

En los anexos 2 y 3 se muestra los códigos VHDL para realizar los diferentes modos de funcionamiento.

En total se ha creado tres modos de funcionamiento, que se explicarán en este apartado.

4.3.4.1 Modo de funcionamiento básico

7	15	23	31	39	47	55	63
6	14	22	30	38	46	54	62
5	13	21	29	37	45	53	61
4	12	20	28	36	44	52	60
3	11	19	27	35	43	51	59
2	10	18	26	34	42	50	58
1	9	17	25	33	41	49	57
0	8	16	24	32	40	48	56

Tabla 2: Modo de funcionamiento básico

En este modo de funcionamiento solo se habilita el par de ROs conectados a los contadores de oscilaciones. Esto se consigue utilizando los mismos contadores utilizados para controlar

los multiplexores. De esta forma utilizando estos dos contadores, seleccionamos dos ROs, los habilitamos y conectamos a los contadores.

4.3.4.2 Modo de funcionamiento habilitación total

7	15	23	31	39	47	55	63
6	14	22	30	38	46	54	62
5	13	21	29	37	45	53	61
4	12	20	28	36	44	52	60
3	11	19	27	35	43	51	59
2	10	18	26	34	42	50	58
1	9	17	25	33	41	49	57
0	8	16	24	32	40	48	56

Tabla 3: Modo de funcionamiento habilitación total

En este modo de funcionamiento se habilitan todos los ROs en el código VHDL de la creación del ROPUF. El funcionamiento es similar al anterior modo, solo que ahora se habilitan todos los ROs, para ver como se ve afectada la frecuencia de oscilación del par de ROs a comparar por la actividad electrónica generada por el resto de ROs.

4.3.4.3 Modo de funcionamiento habilitación de una columna

7	15	23	31	39	47	55	63
6	14	22	30	38	46	54	62
5	13	21	29	37	45	53	61
4	12	20	28	36	44	52	60
3	11	19	27	35	43	51	59
2	10	18	26	34	42	50	58
1	9	17	25	33	41	49	57
0	8	16	24	32	40	48	56

Tabla 4: Modo de funcionamiento habilitación de una columna

En este modo se habilita la columna de ROs de uno de los ROs bajo comparación. Por ejemplo, si se está realizando las comparaciones del RO 19 se habilitan todos los ROs de esa columna y el RO con el que se compara.

4.3.5 ChipScope Pro

ChipScope Pro es una herramienta de xilinx que nos permite integrar componentes de medida y test con el circuito diseñado dentro de la FPGA. Dispone de varios núcleos para integrar con nuestro diseño y poder así ver las señales internas y nodos en la FPGA. Los núcleos de los que dispones son [3]:

- ILA (Integrated Logic Analyzer): con este núcleo se puede monitorizar las señales internas de nuestro diseño.
- VIO (Virtual Input/Output): este núcleo nos permite monitorizar y conducir señales internas en tiempo real
- ICON (Integrated Controller): este núcleo proporciona la comunicación entre el ILA y VIO con la herramienta chipScope Pro Analyzer.

Esta herramienta nos permite capturar y almacenar el valor de las señales de nuestro proyecto mientras se está ejecutando en la FPGA. Para lograr este propósito hay que:

- Utilizar el ChipScope Pro core Inserter y seguir los pasos necesarios para configurar el ILA y poder así elegir las señales de nuestro pryecto que queremos capturar. En nuestro caso serán los ROs y sus respectivos valores de frecuencia de oscilación que se obtienen con los contadores.
- Una vez seleccionadas las señales a capturar, con la herramienta ChipScope pro Analyzer y el código cargado en la FPGA podemos guardar el valor de las señales seleccionadas durante la ejecución del código.

Antes de utilizar el ChipScoce Pro Analyzer, hay que cargar el código en la FPGA. Para ello hay que utilizar la herramienta iMPACT de Xilinx.

4.3.6 Pruebas de caracterización

Las pruebas de caracterización se realizan para medir la frecuencia característica de oscilación de lo ROs en diferentes zonas de la FPGA. Con los resultados de estas pruebas se puede ver la distribución de frecuencias en la FPGA. Se puede ver en qué zonas la frecuencia es más alta y en cuales es más baja, además de averiguar en qué zona la frecuencia es más estable, es decir, en que zona la frecuencia no tiene una tendencia a aumentar o disminuir.

El Objetivo de la caracterización es determinar en qué zona de la FPGA las frecuencias son más estables para posteriormente realizar pruebas en esa zona y poder así determinar la salida del ROPUF.

Para realizar esta prueba, se ha modificado el código VHDL para conectar el mismo RO a los dos contadores de oscilación. Teóricamente ambos contadores deben sacar la misma

frecuencia, pero debido a retardos aleatorios en la señales (como ya se ha explicado anteriormente), la frecuencia de uno será ligeramente mayor que la del otro.

El ChipScope ha sido configurado para obtener 4096 frecuencias. En este caso como se conecta el mismo RO a los dos contadores de oscilación, se obtienen los valores de la frecuencias en las comparaciones de los 64 ROs 64 veces, $64 * 64 = 4096$. Esto hace un total de $2 * 4096 = 8192$ frecuencias de cada RO.

Para obtener las medidas de las frecuencias características de todos los ROs se ha seguido los siguientes pasos:

- Se ha cargado el código VHDL, en uno de los tres modos de funcionamiento, en la FPGA.
- Con la FPGA ejecutando el código, se configura el ChipScope Analyzer para obtener los valores de las frecuencias de todos los ROs
- Una vez obtenidos los valores se guarda en un archivo .prn, se hace reset a la FPGA y se repite el proceso hasta obtener tres archivos .prn.

Este proceso se ha repetido para las cinco zonas en la que se ha divido la FPGA (Figura 18) y para la zona donde se ha realizado el FloorPlan. A su vez este proceso se repite para los tres modos de funcionamiento, obteniendo así los datos de la FPGA en sus diferentes zonas y en diferentes modos de funcionamiento.

El anterior proceso se ha replicado en cuatro placas FPGA Spartan-3E, para determinar sus frecuencias características.

Después de acabar la toma de medidas se habrán obtenido 18 archivos .prn por cada modo de funcionamiento lo que equivale a 54 archivos .prn por cada FPGA. Esta gran cantidad de datos será tratada y analizada posteriormente con la herramienta Matlab para conseguir así caracterizar las frecuencias de cada FPGA.

4.3.7 Pruebas para Registro

Las pruebas de registro se realizan para obtener las medidas de las frecuencias resultantes de todas las comparaciones de los ROs en los diferentes modos y en las diferentes zonas. Estas frecuencias se guardan y sirven como base de datos. Después de la caracterización de las FPGAs y de determinar en qué zona las frecuencias son más estables, se utiliza esta base de datos para obtener todas la comparaciones de ROs de esa zona y estudiar si es posible identificar y autenticar la placa a partir de esas comparaciones, es decir, estudiar si es posible crear una salida de la ROPUF que funcione de forma eficiente.

Para esta prueba se ha modificado el código VHDL para que al ejecutar el código en la Placa FPGA se empieza comparando el RO 0 con todos los ROs incluido el mismo, es decir, primero el RO 0 está conectado a los dos contadores de oscilaciones, después uno de los

contadores se mantiene conectado al RO 0 y el otro se va conectando al resto de ROs cada vez que ha transcurrido el intervalo de tiempo fijado para obtener la oscilación. Esto se repite con el resto de ROs, después de completar las comparaciones del RO 0, se pasa a la comparaciones del RO 1 y así sucesivamente hasta realizar las comparaciones del RO 63. Después de completarse el funcionamiento se obtiene un total de $64 * 64 = 4096$ comparaciones de ROs con sus respectivas frecuencias, que se guardarán.

Para obtener las medidas de las frecuencias resultantes de las comparaciones entre todos los ROs se ha seguido los siguientes pasos:

- Se ha cargado el código VHDL, en uno de los tres modos de funcionamiento, en la FPGA.
- Con la FPGA ejecutando el código, se configura el ChipScope Analyzer para obtener los valores de las frecuencias de todos los ROs
- Una vez obtenidos los valores se guarda en un archivo .prn, se hace reset a la FPGA y se repite el proceso hasta obtener tres archivos .prn.

Este proceso se ha repetido para las cinco zonas en la que se ha divido la FPGA (Figura 18) y en los tres modos de funcionamiento, obteniendo así los datos de la FPGA en sus diferentes zonas y en diferentes modos de funcionamiento.

Después de acabar la toma de medidas se habrán obtenido 15 archivos .prn por cada modo de funcionamiento lo que equivale a 45 archivos .prn por cada FPGA. A partir de estos datos se estudiará posteriormente si es posible obtener una salida del ROPUF que nos sirva para identificar y autenticar las diferentes placas FPGA Spartan-3E.

5 Resultados Experimentales

En este capítulo se tratarán y analizarán los datos obtenidos en las diferentes pruebas. Para tratar los datos se han creado diferentes scripts de Matlab que se explicarán a lo largo de este capítulo.

5.1 Caracterización

El Objetivo de la caracterización es ver la distribución de frecuencias en la FPGA y determinar en qué zona de la FPGA las frecuencias son más estables. Las frecuencias en esta zona serán comparadas entre las diferentes FPGAs y diferentes modos de funcionamiento para ver como se ven afectadas las frecuencias de los ROS y las diferencias existentes.

Para la caracterización, las comparaciones se hacen con el mismo RO, es decir, se conecta el mismo RO a los dos contadores de oscilación. En esta configuración, utilizando ChipScope obtenemos tres archivos .prn con 4096 comparaciones cada uno. El número de ROs utilizado es 64 por lo que obtenemos 128 frecuencias por cada RO. Para evitar posibles influencias del entorno de operación, hemos realizado un reset de la FPGA después de cada adquisición, asegurando así la independencia de las muestras tomadas.

Para tratar estos datos se ha creado un script de Matlab, en el que se cargan dos de los tres archivos .prn obtenidos, se ordenan los datos y se calculan las medias para obtener una única frecuencia media por cada RO. Se calculan las medias para eliminar fluctuaciones aleatorias y minimizar la influencia del entorno en los resultados obtenidos. Después de obtener una única frecuencia media por cada RO, se ordenan las frecuencias de los 64 ROs en una matriz de 8x8. En el anexo 6 se detalla el script creado para tratar los datos.

Este proceso se repite para obtener las matrices 8x8 de todas las zonas y en los diferentes modos de funcionamiento. A partir de estas matrices se realizarán diferentes comparaciones y representaciones para discutir los resultados.

5.1.1 Influencia de la lógica del circuito

Se ha utilizado la herramienta PlanAhead para agrupar la lógica del circuito en un bloque y emplazarlo lejos de la matriz de ROs, con el objetivo de ver como se ven afectadas las

frecuencias de los mismos. Con esto se busca averiguar si la lógica del circuito tiene mucha influencia en las frecuencias, ya que si es así habría que agruparla en una zona alejada de los ROs para minimizar su influencia en las medidas a obtener.

Para hacer esta comprobación se ha utilizado la zona 1. Por lo tanto, para ver si el emplazamiento de la lógica afecta a las frecuencias se ha comprado la matriz 8x8 obtenida en la zona 1 agrupando la lógica del circuito y sin agrupar.

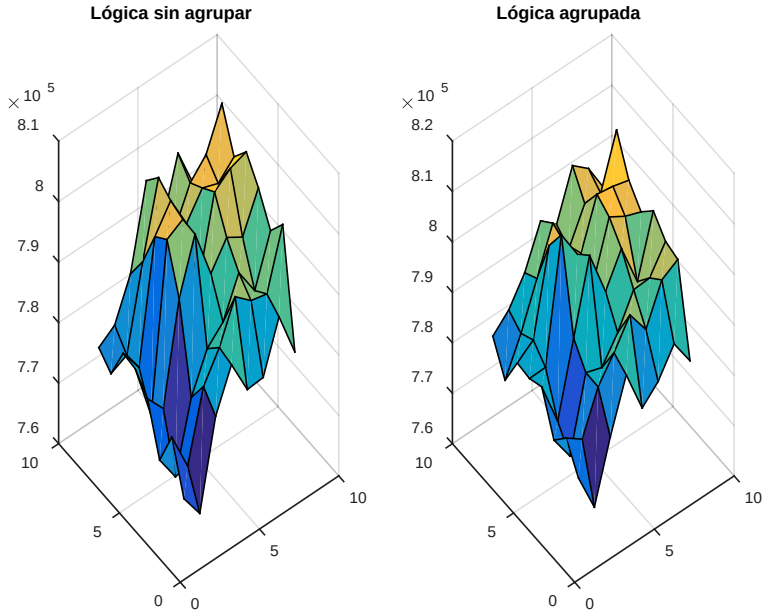

Figura 20: Comparación gráfica entre lógica sin agrupar y agrupada

Para hacer la comparación se ha representado tridimensionalmente las matrices 8x8 de la zona 1 de la lógica agrupada y sin agrupar. En la Figura 20 se muestra la representación tridimensional de las matrices y se puede observar que ambas representaciones son muy similares en forma, por lo que se puede deducir que el emplazamiento de la lógica del circuito no afecta en gran medida a las frecuencias de los ROs y por tanto no sé tendrá en cuenta.

Para reafirmar esta deducción se ha calculado el porcentaje de influencia de la lógica del circuito en la frecuencia de los ROs. Para ello se ha calculado la frecuencia media de las matrices 8x8 con lógica agrupada y sin agrupar.

Después de calcular las frecuencias medias de ambas matrices se obtiene un valor de 786040 (oscilaciones por periodo) para la matriz con lógica sin agrupar y un valor de 787560 (oscilaciones por periodo) para la matriz con lógica agrupada. Con estos valores se concluye que la frecuencia media de los ROs disminuye en un 0,19%. Esta variación se puede considerar despreciable, por lo que no será necesario, para el resto de medidas, realizar un emplazamiento específico para la lógica de control.

5.1.2 Influencia de las zonas del circuito

Para comprobar el impacto del posicionamiento de la matriz (8x8) de ROs en sus frecuencias se ha dividido la FPGA en cinco zonas (Figura 18). Si juntamos las matrices 8x8 de las zonas 1, 2, 3 y 4 en una única matriz de 16x16 siguiendo el orden de la Figura 18 y representamos tridimensionalmente esta matriz se podrá ver de forma aproximada la distribución de frecuencias en la FPGA.

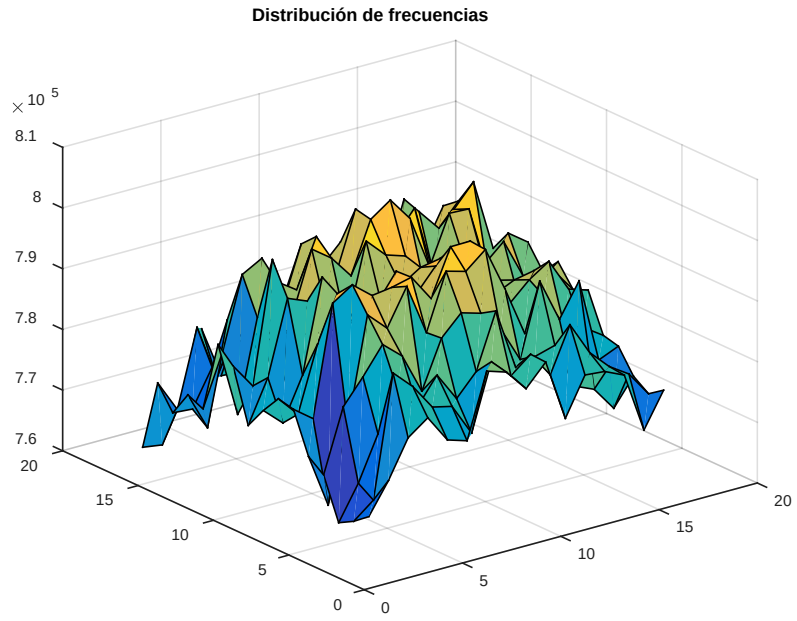

Figura 21: Distribución de frecuencias

En Figura 21 se muestra la representación gráfica de la matriz 16x16 y la distribución de las frecuencias. Se puede observar que en las cuatro zonas hay una tendencia en la que la frecuencia aumenta a medida que nos desplazamos al centro de la FPGA. También se observa que la frecuencia se estabiliza en el centro. Con esto se puede deducir que la frecuencia es más estable en el centro que en los extremos. Para confirmar esta deducción se representa de forma gráfica la matriz 8x8 de la zona 5 situada en el centro de la FPGA.

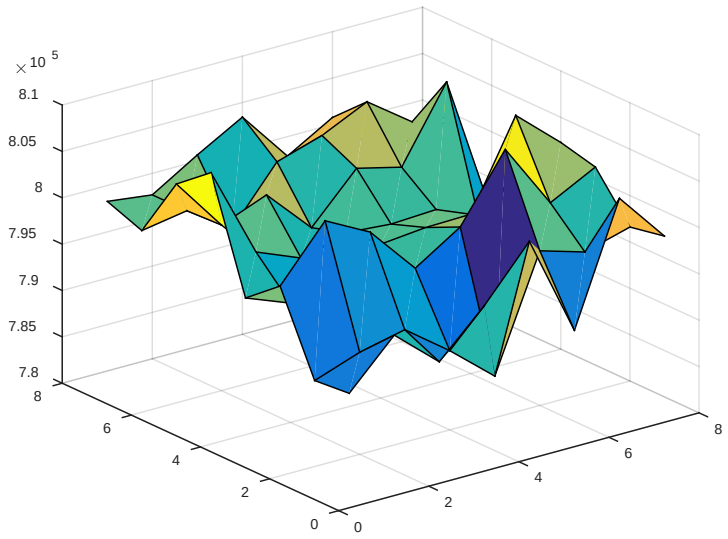

Figura 22: Gráfica de la matriz 8x8 de la zona 5

Observando la representación gráfica de la matriz 8x8 de la zona 5 se puede ver que es mucho más estable que las otras zonas, ya que no hay ninguna tendencia a aumentar o disminuir. Con esto se puede confirmar que en el centro es donde más estable están las frecuencias y que la zona 5 es la zona más estable. Por lo tanto, la zona 5 es la zona óptima para analizar cómo se ven afectadas las frecuencias de los ROs por los diferentes modos de funcionamiento y si es posible diferenciar las Placas FPGA utilizando el ROPUF.

5.1.3 Influencia de la actividad electrónica generada por ROs

Una vez determinada la zona óptima (zona 5) para analizar si es posible diferenciar las placas FPGA, se va estudiar cómo se ven afectadas las frecuencias de los ROs en esta zona por la actividad electrónica generada por ROs. Para ello se va a comparar la matriz 8x8 de la zona 5 entre el modo de funcionamiento básico, el modo de funcionamiento habilitación total y el modo de funcionamiento habilitación de una columna.

Al aumentar la actividad electrónica generada por los ROs, la frecuencia de los ROs disminuye, esta disminución puede afectar a las comparaciones entre ROs y por tanto cambiar la salida del ROPUF. Para comprobar este efecto se representarán y realizarán cálculos con las matrices 8x8 de la zona 5 entre diferentes modos de funcionamiento para ver como disminuyen las frecuencias.

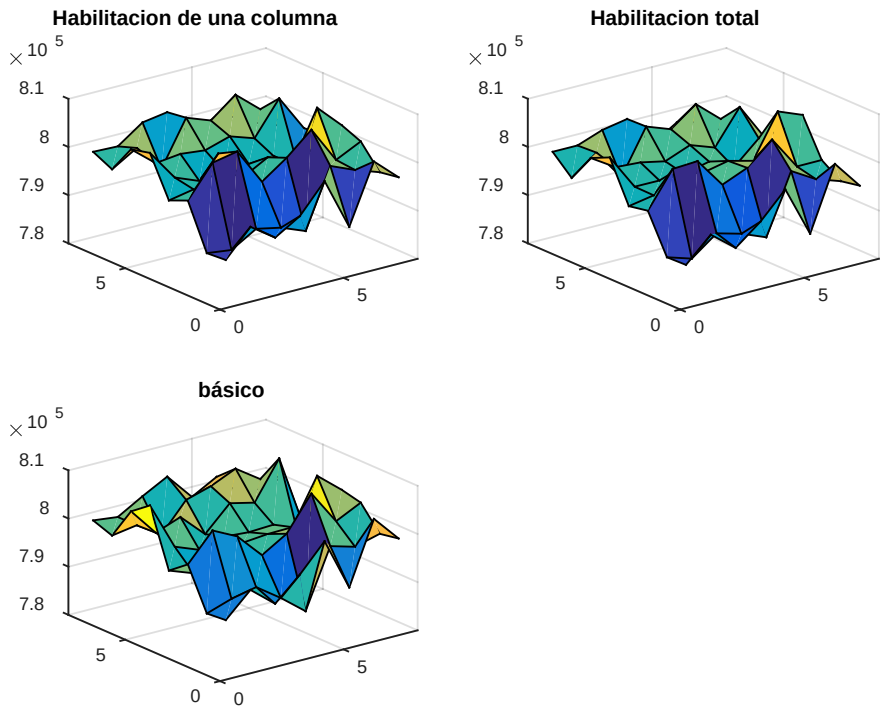

Figura 23: Representación de la matriz 8x8 en la zona 5 en diferentes modos de funcionamiento

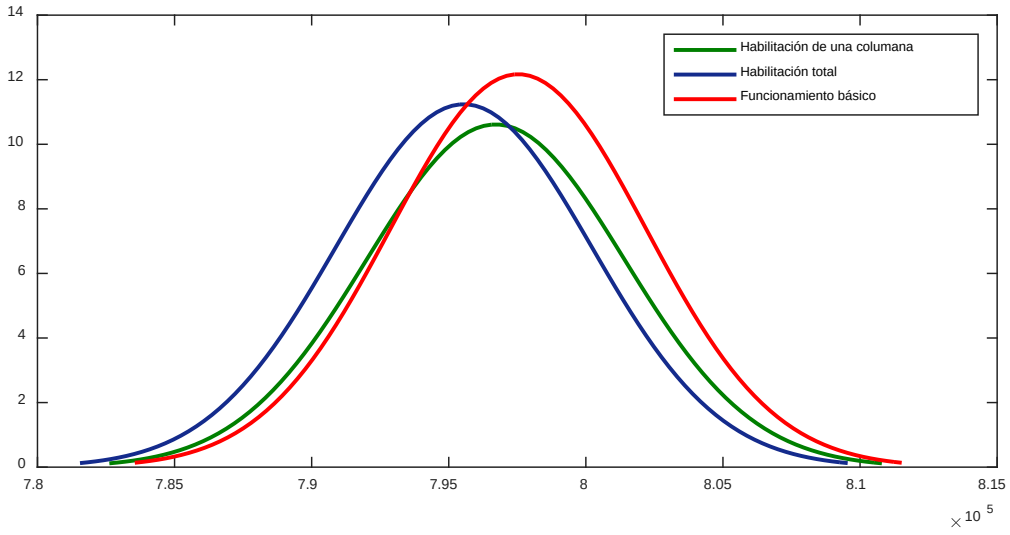

Figura 24: Distribución normal de la matriz 8x8 en los diferentes modos de funcionamiento

Al representar la matriz 8x8 en los diferentes modos de funcionamiento (Figura 23) no se observan mucha diferencia a simple vista, pero si representamos la distribución normal (Figura 24), se observa que una mayor actividad electrónica implica que la curva está más desplazada a la izquierda, es decir, las frecuencias son más bajas.

Para calcular el porcentaje de desplazamiento entre el modo funcionamiento básico y los otros modos, primero se ha calculado la diferencia entre las frecuencias medias de la matriz 8x8 del modo de funcionamiento básico y los otros modos, y después se ha divido esta diferencia entre la diferencia de la frecuencia máxima y mínima del modo de funcionamiento básico.

Después de realizar los cálculos, se ha concluido que la frecuencia de los ROs en el modo de habilitación de una columna está desplazada un 3.73 % a la izquierda respecto a la frecuencia de los ROs en el modo de funcionamiento básico. La frecuencia de los ROs en el modo de habilitación total está desplazada un 8.96 % a la izquierda respecto a la frecuencia de los ROs en el modo de funcionamiento básico.

5.2 Registro

El objetivo del registro es crear una base de datos, con las comparaciones entre todos los ROs. Estas comparaciones se utilizarán para crear la salida del ROPUF y en procesos de identificación y autenticación. Para obtener la salida del ROPUF se compararán dos métodos, que se explicarán a lo largo de este apartado.

Para el registro se ha modificado el código VHDL para realizar comparaciones entre todos los ROs, para ello se compara un RO con el resto de ROs utilizando los contadores de oscilación, una vez se ha comparado con todos los ROs, se cambia a otro para realizar sus comparaciones con el resto de ROs. Este proceso se repite hasta obtener las comparaciones entre todos los ROs. Para guardar las frecuencias de las comparaciones se ha configurado el ChipScope para generar tres archivos .prn, haciendo reset entre cada archivo para eliminar posibles influencias del entorno, con 8192 muestras cada archivo, que equivalen a 4096 frecuencias de cada contador.

Para tratar los datos obtenido se ha creado un script de Matlab para cargar y ordenar los datos del fichero .prn y crear matrices de 64x64 con las comparaciones de las frecuencias. En el Anexo 7 se detalla el script creado para tratar los datos.

5.2.1 Método clásico y método de las diferencias

Método clásico de comparación

En el método clásico de comparación se compran los resultados de los contadores de oscilación, si el resultado de uno es mayor que el otro se genera un ‘1’ y si es menor se genera un ‘0’. Para implementar este método en este trabajo, se comparan las frecuencias de los dos contadores de oscilaciones y según el resultado se genera un ‘1’ o un ‘0’. Con el resultado de estas comparaciones se ha creado una matriz de 64x64 con unos y ceros, quedando como resultado una matriz con las comparaciones entre los 64 ROs.

Método de las diferencias

En este trabajo se propone un método de comparación basado en las diferencias entre las frecuencias de los ROs bajo comparación. Al realizar la comparación del resultado de los contadores de oscilación, en vez de generar un '1' o '0' se calcula la diferencia. Una vez calculadas las diferencias se ordenan en una matriz de 64x64, obteniendo de esta forma las diferencias de las comparaciones entre todos los ROs.

En la Figura 25 se muestra la representación gráfica de la matriz 64x64 de ceros y unos obtenida con el método clásico y en la Figura 26 se muestra la representación gráfica de la matriz 64x64 obtenida con el método de las diferencias. Observando ambas gráficas se puede se puede deducir que le método de las diferencias ofrece varias ventajas sobre el método clásico.

- El método de las diferencias ofrece más información que el método clásico, ya que además de conocer que frecuencia es mayor, también se conoce la diferencia entre ellas, con esto se consigue que sea mucho más difícil que salidas de diferentes ROPUF sean iguales.
- En el método de las diferencias se observa mucha más variabilidad que en el clásico y con ello se consigue reforzar aún más la unicidad de las salida del ROPUF.
- Una de las desventaja del método clásico son los ataques predictivos. Por ejemplo observando la matriz de unos y ceros (Figura 25), se ve que la frecuencia del RO 21 es menor que la del RO 29, por lo tanto se puede predecir que todas las frecuencias menores que la del RO21, también son menores que la frecuencia del RO 29, siguiendo esta lógica se puede crear un modelo matemático que prediga el resultado de futuras comparaciones y poder así obtener la salida del ROPUF. Con el método de la diferencia es más complicado hacer ataques predictivos, ya que además de de predecir qué frecuencia es mayor también se necesita predecir el valor de la diferencia.

5.2.2 Correlación entre diferentes modos de funcionamiento

En las representaciones de las matrices 64x64 de ceros y unos y diferencias en los diferentes modos (Figura 26, Figura 27, Figura 28, Figura 29 y Figura 30), se puede ver como el efecto de la actividad electrónica generada por ROs afecta a la salida del ROPUF. Tanto en el método clásico como el de las diferencias la salida del ROPUF cambia al cambiar de modo. Esto se debe a que al aumentar la actividad electrónica, la frecuencia de los ROs disminuye y esta disminución puede ser suficiente para cambiar el resultado de la comparación y por tanto la salida del ROPUF.

Para ver come cambia la salida del ROPUF, se ha calculado la correlación entre las matrices 64x64 de los diferentes modos de funcionamiento.

Utilizando el método clásico se obtiene una correlación en la Tabla 5.

Modo de funcionamiento	Modo de funcionamiento	Correlación
Modo básico	Modo de habilitación total	0.607
Modo básico	Modo de habilitación de un columna	0.635
Modo de habilitación total	Modo de habilitación de un columna	0.7

Tabla 5: Correlaciones utilizando el método clásico

Utilizando el método de las diferencias se obtiene una correlación en la Tabla 6

Modo de funcionamiento	Modo de funcionamiento	Correlación
Modo básico	Modo de habilitación total	0.712
Modo básico	Modo de habilitación de un columna	0.773
Modo de habilitación total	Modo de habilitación de un columna	0.759

Tabla 6: Correlación utilizando el método de las diferencias

Al calcular la correlación se observa cómo cambia la salida del ROPUF con el modo de funcionamiento. Esto se puede utilizar para calcular una salida de ROPUF combinando los diferentes modos de funcionamiento para conseguir que la salida sea aún más impredecible. Cabe destacar que el hecho de que se obtengan correlaciones más bajas para el método clásico, no implica necesariamente una mejor salida ya que la correlación de esta matriz de unos y ceros favorece la obtención de una menor correlación.

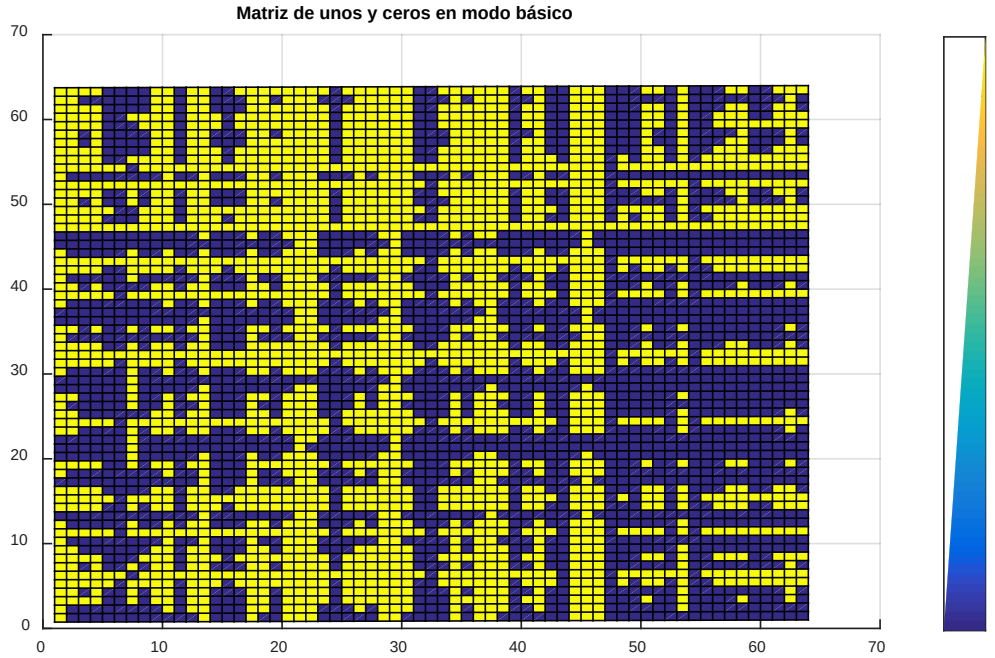

Figura 25: matriz 64x64 de unos y ceros en modo de funcionamiento básico

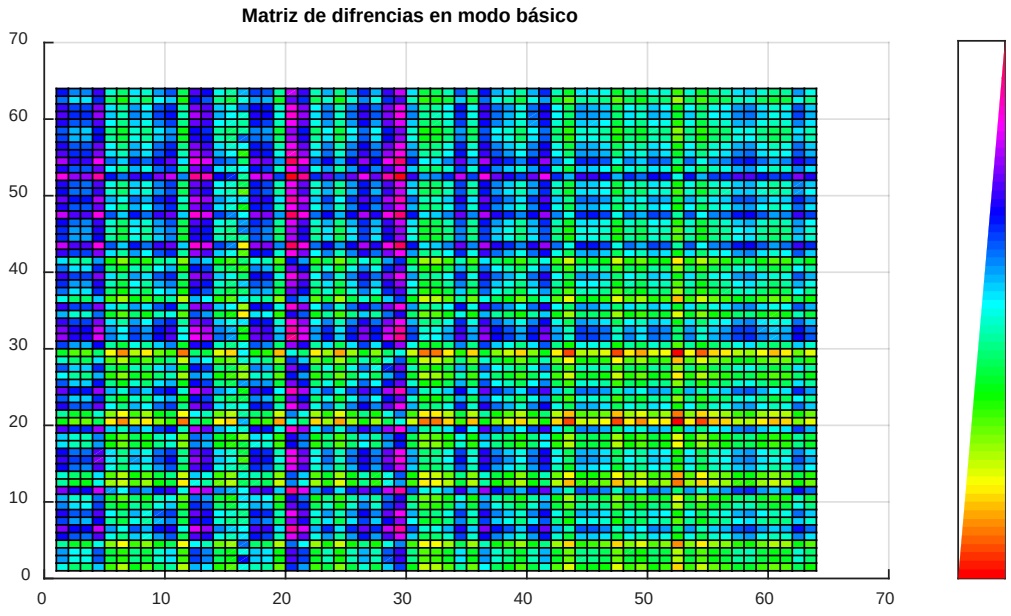

Figura 26: matriz 64x64 de diferencia en valor absoluto en modo de funcionamiento básico

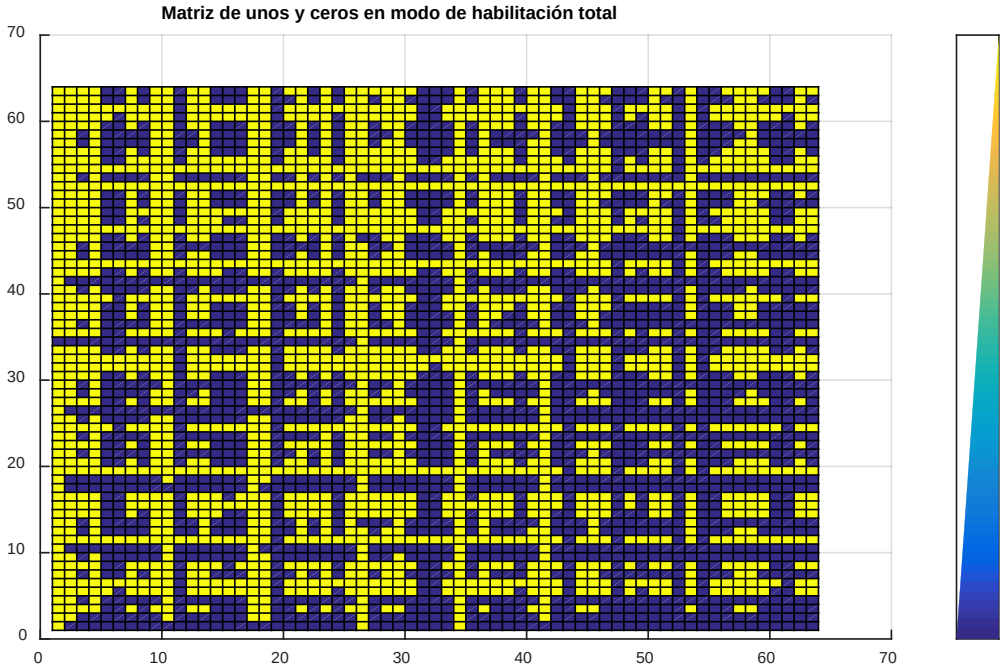

Figura 27: matriz 64x64 de unos y ceros en modo de funcionamiento habilitación total

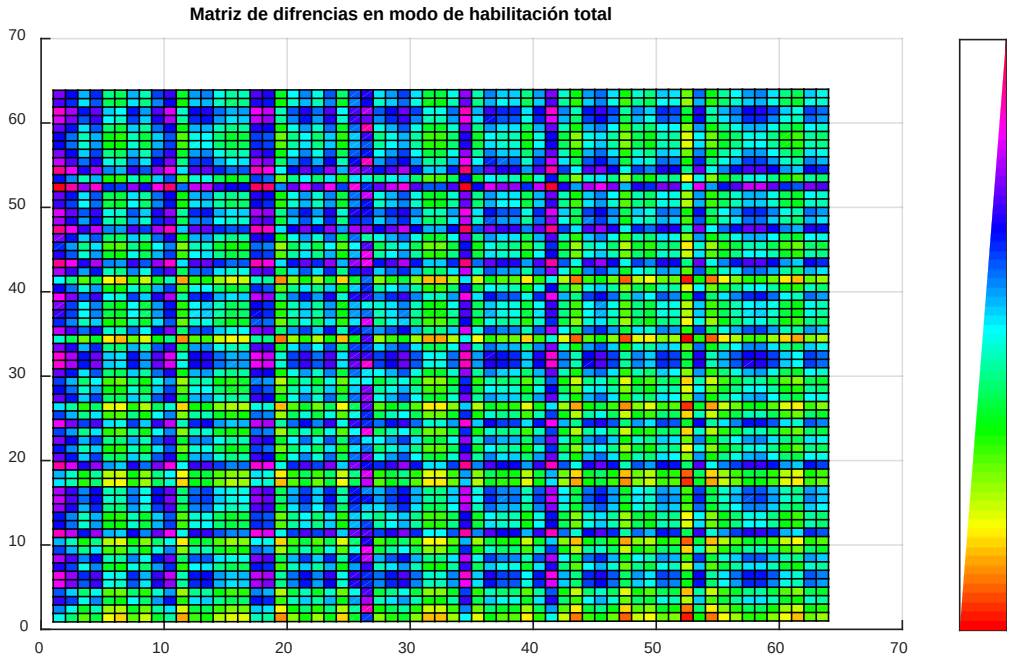

Figura 28: matriz 64x64 de unos y ceros en modo de funcionamiento habilitación total

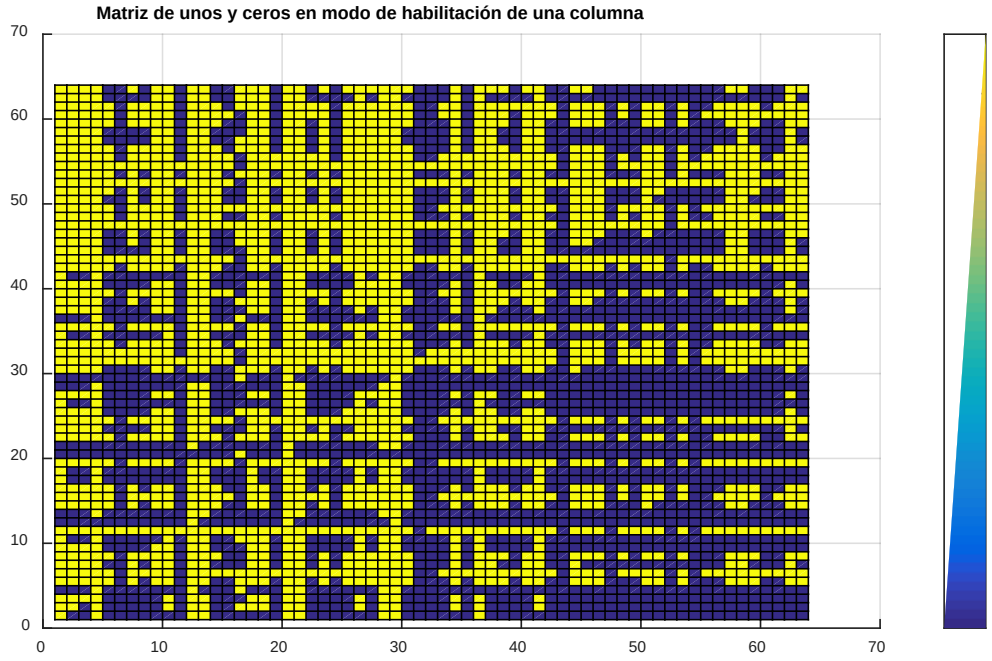

Figura 29: matriz 64x64 de unos y ceros en modo de funcionamiento habilitación de una columna

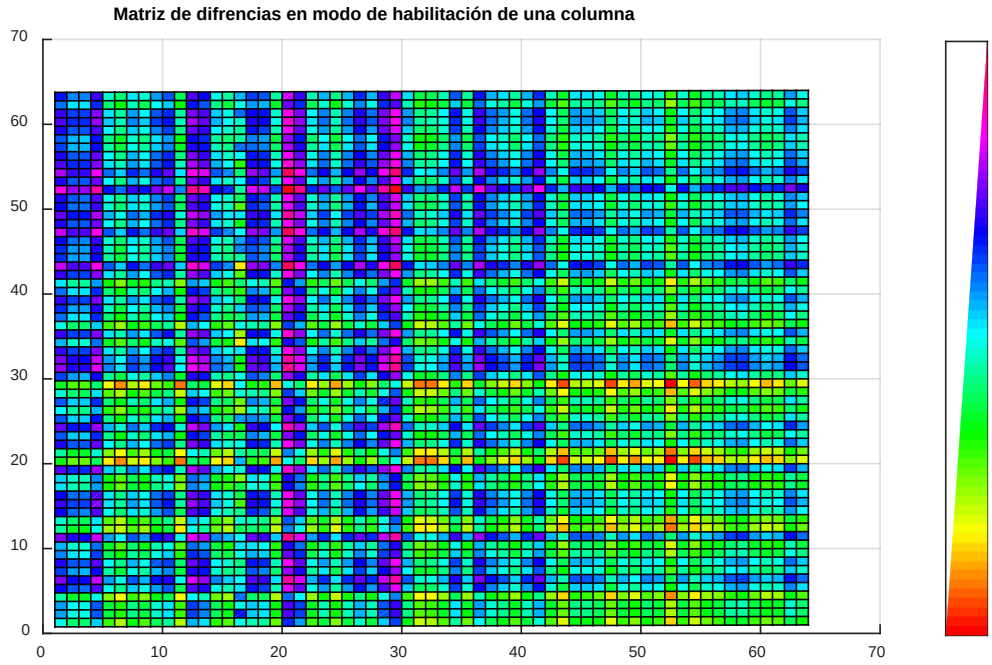

Figura 30: matriz 64x64 de unos y ceros en modo de funcionamiento habilitación de una columna

5.2.3 Correlación entre FPGAs

Para comprobar si es posible identificar de forma diferenciada las FPGAs utilizadas, con las salidas de ROPUF obtenidas, se ha calculado la correlación entre las matrices 64x64 de las cuatro FPGAs utilizadas.

Utilizando el método clásico se obtiene la correlación en Tabla 7.

FPGA	FPGA	Correlación
D289131	D289133	0.406
D289131	D404242	0.348
D289131	D404245	0.373
D289133	D404242	0.328
D289133	D404245	0.373
D404242	D404245	0.1751

Tabla 7: Correlación entre FPGAs utilizando el método clásico

Utilizando el método de las diferencias (en este caso el valor absoluto de la diferencia) se obtiene la correlación en Tabla 8.

FPGA	FPGA	Correlación
D289131	D289133	0.344
D289131	D404242	0.391
D289131	D404245	0.228
D289133	D404242	0.3635
D289133	D404245	0.2282
D404242	D404245	0.0045

Tabla 8: Correlación entre FPGAs utilizando el método de las diferencias

Con los resultados obtenidos, se comprueba que es posible diferenciar las FPGAs con las salidas de ROPUF obtenidas, debido a la baja correlación de las matrices 64x64 entre las diferentes FPGAs.

6 Conclusiones

En este capítulo final se exponen las conclusiones a las que se ha llegado tras la implementación y desarrollo del trabajo fin de grado, así como las líneas de trabajo futuras hacia las que se puede dirigir este proyecto.

6.1 Conclusiones

En este trabajo fin de grado se ha estudiado la respuesta de una ROPUF en diferentes escenarios. Se han considerado diferentes situaciones, donde el emplazamiento de la matriz de ROs o la actividad electrónica generada por diversos componentes del PUF, puedan afectar a la respuesta final.

La FPGA Spartan-3E ha sido utilizada para la obtención de los resultados. Esta FPGA ha sido convenientemente instrumentalizada para poder obtener los datos necesarios para el estudio de la respuesta del ROPUF.

A la vista de los resultados podemos concluir que :

- A la hora de caracterizar la FPGA, la lógica del circuito no tiene una gran influencia en las frecuencias de los ROs, por lo que no es necesario tenerla en cuenta a la hora de calcular la salida ROPUF.
- En cuanto a la influencia de las zonas (Figura 18) de la FPGA en las frecuencias de los ROs, se concluye que en los extremos de la FPGA las frecuencias de los ROs son menores y hay una tendencia en la que la frecuencia aumenta a medida que nos desplazamos al centro de la FPGA y se estabiliza una vez estamos en el centro. También se ha concluido que la zona 5 de la FPGA es la zona más optima para analizar cómo se ven afectadas las frecuencias de los ROs por los diferentes modos de funcionamiento y para calcular la salida del ROPUF.
- Cuando se ha analizado la influencia de la actividad electrónica generada por los ROs en los diferentes modos de funcionamiento, se ha concluido que cuanta más actividad electrónica generada, la frecuencia de los ROs es ligeramente menor, pero suficiente para cambiar el resultado de la salida del ROPUF.

- Después de comparar el modo de comparación clásico con el método de las diferencias, se concluye que el método de la diferencias aporta más información que el clásico, ya que aparte de conocer que frecuencia de RO es mayor, también se conoce la diferencia entre ambas. Además con el método de la diferencia es más difícil realizar ataques predictivos.
- Para finalizar después de calcular la correlación de la matriz de 8x8 de la zona 5 entre diferentes FPGAs, se concluye que es lo suficientemente baja para diferenciar las FPGAs de forma separada.

6.2 Líneas futuras

Las líneas futuras de este trabajo se resumen en los siguientes puntos:

- Es necesario aumentar el número de FPGAs para poder obtener datos concluyentes. A su vez, puede ser interesante la comparación entre diferentes tecnologías de FPGA.
- La inclusión de nuevas métricas (ej, distancia de Hamming) y un análisis estadístico más exhaustivo de los datos pueden ser necesarios a la hora de identificar unívocamente cada una de las FPGAs.
- El desarrollo de un nuevo protocolo de autenticación, que haga uso de los distintos modos de funcionamiento propuestos, nos puede ayudar en la extensión de pares pregunta/respuesta.
- Creación de una página web con una base de datos que contenga las medidas obtenidas para ponerlas a disposición de la comunidad científica.

7 Planificación del trabajo y Presupuesto

En este capítulo se explicará la planificación a seguir para realizar el proyecto así como su duración. También se calculará el presupuesto necesario para realizar el trabajo, en el que se incluirá los costes de los materiales utilizados y el coste del tiempo invertido por el personal.

7.1 Planificación del trabajo

Antes de abordar el trabajo se ha dividido en varias tareas, las cuales serán explicadas a lo largo de este apartado.

- **Documentación**: se ha estudiado en qué consiste el trabajo de fin de grado y las herramientas a utilizar para cumplir los objetivos del mismo.
- **Desarrollo del ROPUF**: consiste en crear los diferentes bloques que conforman el ROPPUF.
- **Pruebas en FPGA**: se realizan todas las pruebas necesarias en FPGAs.
- **Tratamiento de datos**: se tratan los datos obtenidos a partir de las pruebas realizadas para obtener los resultados.
- **Realización de la memoria**: se realiza el documento descriptivo del trabajo realizado.

Tarea	Duración (horas)
Documentación	40
Desarrollo del ROPUF	80
Pruebas en FPGAs	60
Tratamiento de datos	50
Realización de la memoria	90
Total	**320**

Tabla 9: Desglose de tareas

En Tabla 9 se puede observar el desglose de las diferentes tareas y su duración en horas, este desglose se utilizará para calcular el coste del tiempo invertido por el personal.

7.2 Presupuesto

7.2.1 Coste de recursos

Los recursos que han sido requeridos para este trabajo son:

- Un ordenador portátil para utilizar las herramientas necesarias para la creación del ROPUF, el tratamiento de datos, búsqueda de documentación y realización de la memoria.
- Cuatro placas FPGA spartan-3E para realizar las diferentes pruebas necesarias.
- Licencia de la herramienta Xilinx para utilizar diferentes herramientas para la creación del ROPUF y realizar pruebas en las FPGAs. En este caso la licencia es gratuita para estudiantes.
- Licencia de la herramienta Matlab para el tratamiento de los datos de las pruebas realizadas.

Recurso	Cantidad (ud)	Coste (€)
Hp Pavilion g6	1	450
Spartan-3E	4	800
ISE Design Suit	1	0
Matlab Student Suite	1	69
Total		**1319**

Tabla 10: Desglose del coste de recursos

En Tabla 10 se puede ver que el coste de recursos asciende a un total de 1319 €.

7.2.2 Coste de personal

Para este trabajo es necesario a un Ingeniero para realizar las tareas indicadas en el desglose de tareas (Tabla 9), y a un jefe de proyecto para realizar la supervisión del trabajo.

Recurso	Salario (€/hora)	Tiempo (horas)	Coste (€)
Ingeniero junior	12	320	3840
Jefe de proyecto	15	30	450
Total		**350**	**4290**

Tabla 11: Desglose del coste de personal

En Tabla 11 se puede ver que el coste de personal asciende a un total de 4290 €.

7.2.3 Coste total

Concepto	Precio (€)
Coste de recursos	1319
Coste de personal	4290
Coste indirecto (20 %)	1121.8
Subtotal	6730.8
IVA (21 %)	1211.54
Total	**7942.34**

Tabla 12: Desglose del coste total

En Tabla 12 se puede ver que el coste total asciende a un total de 7942.34 €.

Anexo 1. Código VHDL del Ring Oscillator

```
--- Código Para la creación del Ring Oscillatot ---

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Macro is
    Port ( En : in  STD_LOGIC;
           Q1 : out  STD_LOGIC);
end Macro;

architecture Behavioral of Macro is

attribute keep: string;
Signal Q2: std_logic_vector(5 downto 0) := (others=>'0');
attribute keep of Q2: signal is "true";

begin
-- Proceso para crear el circuito del Ring Oscilator
-- con un AND y cinco not
Process(en,q2)
begin
      q2(0) <= en and q2(5);  -- habilitación del RO con señal en
      q2(1) <= not q2(0);
      q2(2) <= not q2(1);
      q2(3) <= not q2(2);
      q2(4) <= not q2(3);
      q2(5) <= not q2(4);
      q1 <= q2(5);            -- realimentación de la salida
end process;

end Behavioral;
```

Anexo 2. Código VHDL para caracterización

```
--- Código para la creación del ROPUF para la caracterización ---

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.STD_LOGIC_arith.ALL;

entity top is
port (clk: in std_logic;
            Reset: in std_logic);
end top;

architecture Behavioral of top is

-- blouqe de RO creado con la Hard Macro
component RO_reducida port ( En : in std_logic; Q1: out std_logic);
end component;

Signal cuenta :integer range 0 to 262144; -- timepo fijado para contar las
oscilaciones
signal flag: std_logic; -- señal que indica que el periodo de tiempo fijado
ya ha transcurrido
signal pos1: integer range 0 to 63; -- equivale al número de RO en la
matriz de ROs
signal RO_A,RO_B,contador_A, contador_B: std_logic_vector (31 downto 0); --
para contar las oscilaciones de los ROs
Signal Qx: std_logic_vector(63 downto 0); -- salidas de los ROs
signal X,Y: std_logic;  -- señales que marcan el paso para contar las
oscilacones de los ROs
Signal Enx: std_logic_vector(63 downto 0);      -- señal de habilitacion de
los ROs

attribute keep: string;
attribute keep of RO_A,RO_B: signal is "true";

begin
-- Replica del RO 64 veces
puf_gen: for i in 0 to 63 generate
pufx: RO_reducida port map(Enx(i), Qx(i));
end generate puf_gen;
```

```
-- Proceso para fijar el periodo de tiempo de oscilacion
Process(clk, reset)
	begin
	if reset='1' then
		cuenta<=0;
	elsif clk' event and clk='1' then
		if cuenta =262144 then
			cuenta<=0;
		else
			cuenta<= cuenta+1;
		end if;
	end if;
end process;

-- Proceso para activar señal que indica que el periodo de tiempo fijado ya
ha transcurrido
Process(clk, reset)
	begin
	if reset='1' then
		flag<='0'; --señal que indica que el periodo de tiempo fijado
ya ha transcurrido
	elsif clk' event and clk='1' then
		if cuenta=(262144-1) then
			flag<='1';
		else
			flag<='0';
		end if;
	end if;
end process;

-- para elegir las hard macro a activar
Process(clk, reset)
	begin
	if reset='1' then
		pos1<=0;	-- pos1 equivalen al número del RO a activar
	elsif clk' event and clk='1' then
		if flag='1' then
			if pos1 =63 then
				pos1<=0;
			else
				pos1<= pos1+1;
			end if;
		end if;
	end if;
end process;

X<=Qx(pos1);
Y<=Qx(pos1);

-- contador de oscilaciones A
Process(reset,X)
	begin
	if reset='1' then
		contador_A<=(others => '0');
	elsif X' event and X='1' then
		if flag='1' then
			contador_A<= (others => '0');
```

```
            else
                  contador_A<=contador_A + 1;
            end if;
      end if;
end process;

-- Proceso para guardar las oscilaciones del RO A contadas durante el
periodo de tiempo fijado
Process(reset,clk)
      begin
      if reset='1' then
            RO_A<=(others => '0'); -- señal para almacenar el número de
oscilaciones del RO A
      elsif clk' event and clk='1' then
            if cuenta=(262144-1) then
                  RO_A<=contador_A;
            end if;
      end if;
end process;

-- Contador de oscilaciones B
Process(reset,Y)
      begin
      if reset='1' then
            contador_B<=(others => '0');
      elsif Y' event and Y='1' then
            if flag='1' then
                  contador_B<= (others => '0');
            else
                  contador_B<=contador_B + 1;
            end if;
      end if;
end process;

-- Proceso para guardar las oscilaciones del RO B contadas durante el
periodo de tiempo fijado
Process(reset,clk)
      begin
      if reset='1' then
            RO_B<=(others => '0');  -- señal para almacenar el número de
oscilaciones del RO B
      elsif clk' event and clk='1' then
            if cuenta=(262144-1) then
                  RO_B<=contador_B;
            end if;
      end if;
end process;

--- Modos de funcinamiento ---

-- A continuación se han realizado tres modos de funcionamiento
-- El código está comentado, descomentar el modo deseado

-- Modo de funcionamiento básico --
-- En este modo solo se habilitan los ROs a comparar

-- Process (clk,reset)
```

```
      -- Begin
      -- if reset='1' then
            -- Enx<=(others=>'0');
      -- elsif clk' event and clk='1' then
            -- Enx<=(others=>'0');
            -- Enx(pos1)<='1';
      -- end if;
-- end Process;

-- Modo de funcionamiento habilitación de una columna --
-- En este modo se habilita la columna de ROs del RO bajo comparación

-- Process (clk,reset)
      -- Begin
      -- if reset='1' then
            -- Enx<=(others=>'0');
            -- Enx(7 downto 0)<="11111111";
      -- elsif clk' event and clk='1' then
                  -- Enx<=(others=>'0');
                  -- if (pos1<=7) then
                        -- Enx(7 downto 0)<="11111111";
                  -- elsif (pos1 >7 and pos1<=15) then
                        -- Enx(15 downto 8)<="11111111";
                  -- elsif (pos1>15 and pos1<=23) then
                        -- Enx(23 downto 16)<="11111111";
                  -- elsif (pos1>23 and pos1<=31)then
                        -- Enx(31 downto 24)<="11111111";
                  -- elsif (pos1>31 and pos1<=39) then
                        -- Enx(39 downto 32)<="11111111";
                  -- elsif (pos1>39 and pos1<=47) then
                        -- Enx(47 downto 40)<="11111111";
                  -- elsif (pos1>47 and pos1<=55) then
                        -- Enx(55 downto 48)<="11111111";
                  -- elsif (pos1>55 and pos1<=63) then
                        -- Enx(63 downto 56)<="11111111";
                  -- end if;
      -- end if;
-- end process;

-- Modo de funcionamiento habilitación total --
-- En este modo se habilitan todos los ROs

-- Enx<=(others => '1');

end Behavioral;
```

Anexo 3. Código VHDL para Registro

```
--- Código para la creación del ROPUF para el registro ---

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;
use IEEE.STD_LOGIC_arith.ALL;



entity top is
port (clk: in std_logic;
            Reset: in std_logic);
end top;

architecture Behavioral of top is

-- blouqe de RO creado con la Hard Macro
component RO_reducida port ( En : in std_logic; Q1: out std_logic);
end component;

Signal cuenta :integer range 0 to 262144; -- timepo fijado para contar las
oscilaciones
signal flag: std_logic; -- señal que indica que el periodo de tiempo fijado
ya ha transcurrido
signal pos1,pos2: integer range 0 to 63; -- equivalen al número de RO en la
matriz de ROs
signal RO_A,RO_B,contador_A, contador_B: std_logic_vector (31 downto 0); --
para contar las oscilaciones de los ROs
Signal Qx: std_logic_vector(63 downto 0); -- salidas de los ROs
signal X,Y: std_logic;  -- señales que marcan el paso para contar las
oscilacones de los ROs
Signal Enx: std_logic_vector(63 downto 0);      -- señal de habilitacion de
los ROs

attribute keep: string;
attribute keep of RO_A,RO_B: signal is "true";

begin
-- Replica del RO 64 veces
puf_gen: for i in 0 to 63 generate
pufx: RO_reducida port map(Enx(i), Qx(i));
end generate puf_gen;
```

```
-- Proceso para fijar el periodo de tiempo de oscilacion
Process(clk, reset)
	begin
	if reset='1' then
		cuenta<=0;
	elsif clk' event and clk='1' then
		if cuenta =262144 then
			cuenta<=0;
		else
			cuenta<= cuenta+1;
		end if;
	end if;
end process;

-- Proceso para activar señal que indica que el periodo de tiempo fijado ya
ha transcurrido
Process(clk, reset)
	begin
	if reset='1' then
		flag<='0'; --señal que indica que el periodo de tiempo fijado
ya ha transcurrido
	elsif clk' event and clk='1' then
		if cuenta=(262144-1) then
			flag<='1';
		else
			flag<='0';
		end if;
	end if;
end process;

-- Proceso para elegir los ROs a comparar
Process(clk, reset)
	begin
	if reset='1' then
		pos1<=0; pos2<=0; -- pos1 y pos2 equivalen al número del RO a
activar
	elsif clk' event and clk='1' then
		if flag='1' then
			if pos1 =63 then
				pos1<=0;
			else
				pos1<= pos1+1;
			end if;
			if pos2 =63 and pos1=63 then
				pos2<=0;
			elsif pos1=63 then
				pos2<=pos2+1;
			end if;
		end if;
	end if;
end process;

X<=Qx(pos1);
Y<=Qx(pos2);

-- contador de oscilaciones A
Process(reset,X)
```

```
      begin
      if reset='1' then
            contador_A<=(others => '0');
      elsif X' event and X='1' then
            if flag='1' then
                  contador_A<= (others => '0');
            else
                  contador_A<=contador_A + 1;
            end if;
      end if;
end process;

-- Proceso para guardar las oscilaciones del RO A contadas durante el
periodo de tiempo fijado
Process(reset,clk)
      begin
      if reset='1' then
            RO_A<=(others => '0'); -- señal para almacenar el número de
oscilaciones del RO A
      elsif clk' event and clk='1' then
            if cuenta=(262144-1) then
                  RO_A<=contador_A;
            end if;
      end if;
end process;

-- Contador de oscilaciones B
Process(reset,Y)
      begin
      if reset='1' then
            contador_B<=(others => '0');
      elsif Y' event and Y='1' then
            if flag='1' then
                  contador_B<= (others => '0');
            else
                  contador_B<=contador_B + 1;
            end if;
      end if;
end process;

-- Proceso para guardar las oscilaciones del RO B contadas durante el
periodo de tiempo fijado
Process(reset,clk)
      begin
      if reset='1' then
            RO_B<=(others => '0');  -- señal para almacenar el número de
oscilaciones del RO B
      elsif clk' event and clk='1' then
            if cuenta=(262144-1) then
                  RO_B<=contador_B;
            end if;
      end if;
end process;

--- Modos de funcinamiento ---

-- A continuación se han realizado tres modos de funcionamiento
```

```
-- El código está comentado, descomentar el modo deseado

-- Modo de funcionamiento básico --
-- En este modo solo se habilitan los ROs a comparar

-- Process (clk,reset)
      -- Begin
      -- if reset='1' then
            -- Enx<=(others=>'0');
      -- elsif clk' event and clk='1' then
            -- Enx<=(others=>'0');
            -- Enx(pos1)<='1';
            -- Enx(pos2)<='1';
      -- end if;
-- end Process;

-- Modo de funcionamiento habilitación de una columna --
-- En este modo se habilita la columna de ROs de uno de los ROs bajo
comparación

-- Process (clk,reset)
      -- Begin
      -- if reset='1' then
            -- Enx<=(others=>'0');
            -- Enx(7 downto 0)<="11111111";
      -- elsif clk' event and clk='1' then

                  -- Enx<=(others=>'0');
                  -- Enx(pos1)<='1';
                  -- if (pos2<=7) then
                        -- Enx(7 downto 0)<="11111111";
                  -- elsif (pos2 >7 and pos2<=15) then
                        -- Enx(15 downto 8)<="11111111";
                  -- elsif (pos2>15 and pos2<=23) then
                        -- Enx(23 downto 16)<="11111111";
                  -- elsif (pos2>23 and pos2<=31)then
                        -- Enx(31 downto 24)<="11111111";
                  -- elsif (pos2>31 and pos2<=39) then
                        -- Enx(39 downto 32)<="11111111";
                  -- elsif (pos2>39 and pos2<=47) then
                        -- Enx(47 downto 40)<="11111111";
                  -- elsif (pos2>47 and pos2<=55) then
                        -- Enx(55 downto 48)<="11111111";
                  -- elsif (pos2>55 and pos2<=63) then
                        -- Enx(63 downto 56)<="11111111";
                  -- end if;
      -- end if;
-- end process;

-- Modo de funcionamiento habilitación total --
-- En este modo se habilitan todos los ROs

-- Enx<=(others => '1');

end Behavioral;
```

Anexo 4. Código C++

```
/*** Código C++ para generar contenido del archivo UCF  ***/
#include <iostream>
#include <stdio.h>
using namespace std;

/* en este código, se especifica en las variables const_x y const_y
   donde se desea empezar el emplazamiento de los ROs y a partir de esta
   informacón se genera el contenido del archivo UCF con el emplazamiento
   correcto de los  ROs  */

int main ()
{
  int i=0, n=0, a=0, const_x=26, const_y=23, x,y;
  y=const_y;
  x=const_x;
  for (i;i<8;i++)
  {
      for (n;n<8;n++)
      {
          cout <<"INST \"puf_gen[" <<a <<"].pufx\" Loc=SLICE_X"
               <<x <<"Y" <<y <<" ;\n";
          a++;
          y=y+6;
      }
      n=0;
      y=const_y;
      if (x==2 || x==54)
      {
         x=x+10;
      }
      else
      {
          x=x+2;
      }
  }
 getchar();
}
```

Anexo 5. Código del archivo UCF

```
#INST "puf" Loc=SLICE_X32Y69 ;

NET "Clk" TNM_NET = "Clk";
TIMESPEC TS_Clk = PERIOD "Clk" 20 ns HIGH 50 %;

NET "clk" LOC = C9;
NET "Reset" LOC = L13;
NET "Reset" PULLDOWN;

INST "puf_gen[0].pufx" Loc=SLICE_X26Y23 ;
INST "puf_gen[1].pufx" Loc=SLICE_X26Y29 ;
INST "puf_gen[2].pufx" Loc=SLICE_X26Y35 ;
INST "puf_gen[3].pufx" Loc=SLICE_X26Y41 ;
INST "puf_gen[4].pufx" Loc=SLICE_X26Y47 ;
INST "puf_gen[5].pufx" Loc=SLICE_X26Y53 ;
INST "puf_gen[6].pufx" Loc=SLICE_X26Y59 ;
INST "puf_gen[7].pufx" Loc=SLICE_X26Y65 ;
INST "puf_gen[8].pufx" Loc=SLICE_X28Y23 ;
INST "puf_gen[9].pufx" Loc=SLICE_X28Y29 ;
INST "puf_gen[10].pufx" Loc=SLICE_X28Y35 ;
INST "puf_gen[11].pufx" Loc=SLICE_X28Y41 ;
INST "puf_gen[12].pufx" Loc=SLICE_X28Y47 ;
INST "puf_gen[13].pufx" Loc=SLICE_X28Y53 ;
INST "puf_gen[14].pufx" Loc=SLICE_X28Y59 ;
INST "puf_gen[15].pufx" Loc=SLICE_X28Y65 ;
INST "puf_gen[16].pufx" Loc=SLICE_X30Y23 ;
INST "puf_gen[17].pufx" Loc=SLICE_X30Y29 ;
INST "puf_gen[18].pufx" Loc=SLICE_X30Y35 ;
INST "puf_gen[19].pufx" Loc=SLICE_X30Y41 ;
INST "puf_gen[20].pufx" Loc=SLICE_X30Y47 ;
INST "puf_gen[21].pufx" Loc=SLICE_X30Y53 ;
INST "puf_gen[22].pufx" Loc=SLICE_X30Y59 ;
INST "puf_gen[23].pufx" Loc=SLICE_X30Y65 ;
INST "puf_gen[24].pufx" Loc=SLICE_X32Y23 ;
INST "puf_gen[25].pufx" Loc=SLICE_X32Y29 ;
INST "puf_gen[26].pufx" Loc=SLICE_X32Y35 ;
INST "puf_gen[27].pufx" Loc=SLICE_X32Y41 ;
INST "puf_gen[28].pufx" Loc=SLICE_X32Y47 ;
INST "puf_gen[29].pufx" Loc=SLICE_X32Y53 ;
INST "puf_gen[30].pufx" Loc=SLICE_X32Y59 ;
INST "puf_gen[31].pufx" Loc=SLICE_X32Y65 ;
```

```
INST "puf_gen[32].pufx" Loc=SLICE_X34Y23 ;
INST "puf_gen[33].pufx" Loc=SLICE_X34Y29 ;
INST "puf_gen[34].pufx" Loc=SLICE_X34Y35 ;
INST "puf_gen[35].pufx" Loc=SLICE_X34Y41 ;
INST "puf_gen[36].pufx" Loc=SLICE_X34Y47 ;
INST "puf_gen[37].pufx" Loc=SLICE_X34Y53 ;
INST "puf_gen[38].pufx" Loc=SLICE_X34Y59 ;
INST "puf_gen[39].pufx" Loc=SLICE_X34Y65 ;
INST "puf_gen[40].pufx" Loc=SLICE_X36Y23 ;
INST "puf_gen[41].pufx" Loc=SLICE_X36Y29 ;
INST "puf_gen[42].pufx" Loc=SLICE_X36Y35 ;
INST "puf_gen[43].pufx" Loc=SLICE_X36Y41 ;
INST "puf_gen[44].pufx" Loc=SLICE_X36Y47 ;
INST "puf_gen[45].pufx" Loc=SLICE_X36Y53 ;
INST "puf_gen[46].pufx" Loc=SLICE_X36Y59 ;
INST "puf_gen[47].pufx" Loc=SLICE_X36Y65 ;
INST "puf_gen[48].pufx" Loc=SLICE_X38Y23 ;
INST "puf_gen[49].pufx" Loc=SLICE_X38Y29 ;
INST "puf_gen[50].pufx" Loc=SLICE_X38Y35 ;
INST "puf_gen[51].pufx" Loc=SLICE_X38Y41 ;
INST "puf_gen[52].pufx" Loc=SLICE_X38Y47 ;
INST "puf_gen[53].pufx" Loc=SLICE_X38Y53 ;
INST "puf_gen[54].pufx" Loc=SLICE_X38Y59 ;
INST "puf_gen[55].pufx" Loc=SLICE_X38Y65 ;
INST "puf_gen[56].pufx" Loc=SLICE_X40Y23 ;
INST "puf_gen[57].pufx" Loc=SLICE_X40Y29 ;
INST "puf_gen[58].pufx" Loc=SLICE_X40Y35 ;
INST "puf_gen[59].pufx" Loc=SLICE_X40Y41 ;
INST "puf_gen[60].pufx" Loc=SLICE_X40Y47 ;
INST "puf_gen[61].pufx" Loc=SLICE_X40Y53 ;
INST "puf_gen[62].pufx" Loc=SLICE_X40Y59 ;
INST "puf_gen[63].pufx" Loc=SLICE_X40Y65 ;
```

Anexo 6. Script para caracterización

```
% Script para oredenar los datos obtenidos de las pruebas de
caracterización

ROA=0;
ROB=0;
media_A=0;
media_B=0;
media_final=0;
Mat_1=0;
Mat_2=0;
Matriz8x8=0;
fichero=0;
datos=0;
datos_1=0;
datos_2=0;

% carga de ficheros .prn
fichero='Misma_posicion_5-1.prn';
datos=importdata(fichero);
datos_1=sortrows(datos.data,3);

fichero='Misma_posicion_5-2.prn';
datos=importdata(fichero);
datos_2=sortrows(datos.data,3);

% Primer fichero
% Frecuencias del Ring Oscilator A
d=1;
for i=1:64
    for a=1:64
    ROA(a,i)=datos_1(d,4); % matriz 64x64 con 64 frecuencias de cada RO
    d=d+1;
    end
end
media_A=mean(ROA); % media de las 64 frecuencias de cada RO

% Frecuencias del Ring Oscilator B
d=1;
for i=1:64
    for a=1:64
    ROB(a,i)=datos_1(d,5); % matriz 64x64 con 64 frecuencias de cada RO
    d=d+1;
```

```
    end
end
media_B=mean(ROB); % media de las 64 frecuencias de cada RO
media_final=(media_A+media_B)/2; % media de los contadores de oscilacion

% matriz 8x8 con las frecuencias medias de cada RO del fichero 1
b=1
for i=1:8
    for a=8:-1:1
        Mat_1(a,i)=media_final(b)
        b=b+1
    end
end

% Segundo fichero
% Frecuencias del Ring Oscilator A
d=1;
for i=1:64
    for a=1:64
    ROA(a,i)=datos_2(d,4);%RO_A
    d=d+1;
    end
end
media_A=mean(ROA);

% Frecuencias del Ring Oscilator B
d=1;
for i=1:64
    for a=1:64
    ROB(a,i)=datos_2(d,5);
    d=d+1;
    end
end
media_B=mean(ROB); % media de las 64 frecuencias de cada RO
media_final=(media_A+media_B)/2; % media de los contadores de oscilacion

% matriz 8x8 con las frecuencias medias de cada RO del fichero 2
b=1
for i=1:8
    for a=8:-1:1
        Mat_2(a,i)=media_final(b)
        b=b+1
    end
end
Matriz8x8=(Mat_1+Mat_2)/2; % media de los dos ficheros

disp ('Matriz final 1')
disp (Mat_1)
disp ('Matriz final 2')
disp (Mat_2)
disp ('Matriz 8x8 Resultante')
disp (Matriz8x8)
figure('Name','Mapa De Frequencias','NumberTitle','off')
surf(Matriz8x8) % representacion tridimensional de los datos obtenido
saveas(gcf,'Mapa_Frequencias','fig')
save('mat8x8.mat','Matriz8x8')
disp ('------ Fin Del Programa ------')
```

Anexo 7. Script para registro

```
% Script para oredenar los datos obtenidos de las pruebas de registro

% Importación de datos
fichero='Normal_5-2.prn';
datos=importdata(fichero);
datos_1=sortrows(datos.data,4);

% calculo de matriz 64x64 de ceros y uno y diferencias
i=0;
a=0;
d=1;
for i=1:64
    for a=1:64
    diferencia(a,i)=datos_1(d,5)-datos_1(d,6);% matriz de diferencias
    if datos_1(d,5)>=datos_1(d,6)
        comp=1;
    else
        comp=0;
    end;
    unos(a,i)=comp; % matriz de cero y unos
    d=d+1;
    end;
end;

save('diferencia_31_5-2.mat','diferencia');
save('unos_31_5-2.mat','unos');
figure;
surf(abs(diferencia)); % representación grádica de la matriz 64x64 de
diferencias
title('diferencias');
figure;
surf(unos); % representación grádica de la matriz 64x64 de ceros y unos
title('unos y ceros')
disp ('------ Fin Del Programa ------')
```

Anexo 8. Creación de la Hard Macro

Para emplazar el ring oscillator en diferentes zonas de la FPGA, se ha tenido que crear una hard macro, un bloque del RO.

Para ello, en la herramienta Xilinx, vamos al **Process menu**, expandimos la opción **Place &Route options** y abrimos el FPGA Editor (**View/Edit Routed Design**) (Figura 31).

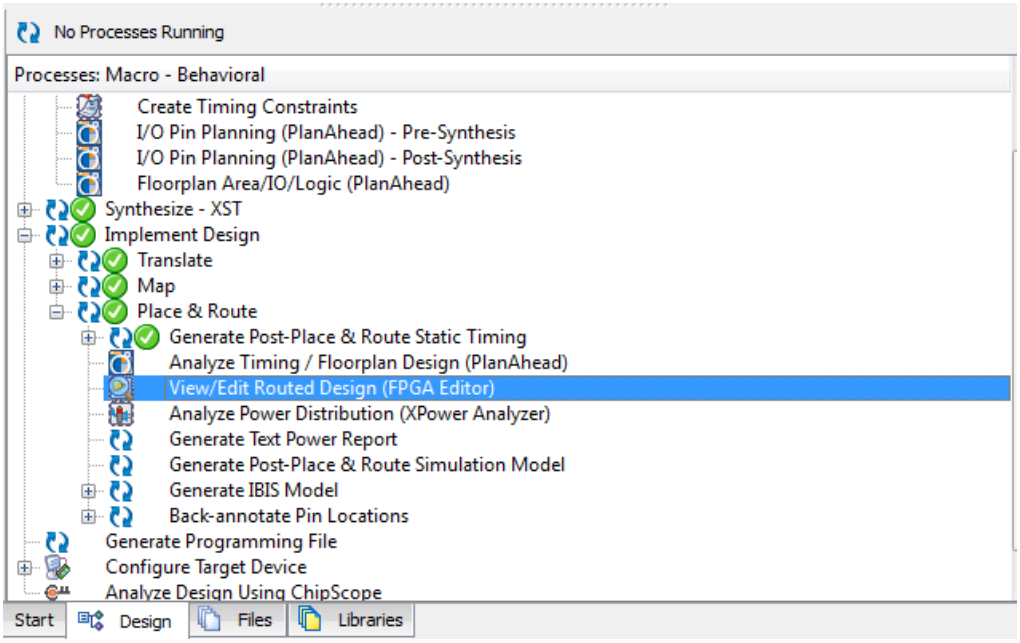

Figura 31: FPGA Editor

En el FPGA Editor, lo primero es gaurdar el diseño como hard macro, para ello vamos **File/Save As** y cuando salta el recuadro seleccionamos la opción **Hard Macro** para guardarlo en estenxión .nmc (Figura 32).

Antes de realizar ningún cambio hay que habilitar el modo **Read Write**, en **File/Main Properties** (Figura 33)**.**

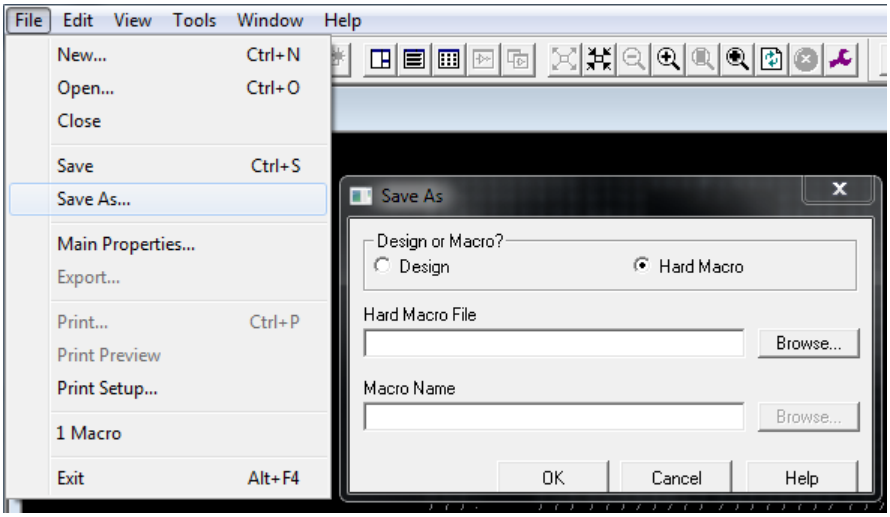

Figura 32: FPGA Editor. Guardar como Hard Macro

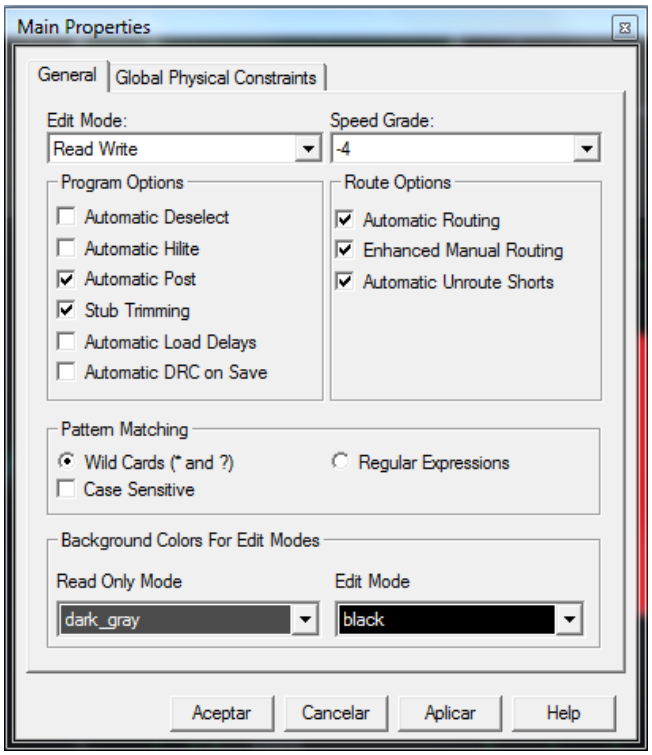

Figura 33: FGPA Editor. Habilitar modo Read Write

Es necesario quitar del diseño todos los componentes que no sean SLICE, como las señales de entrada y salida, para que solo queden solo los inversores y la puerta AND. Para ello, en la ventana de componentes, seleccionamos el componente y en la ventana del esquemático pulsamos botón derecho sobre el componente y seleccionamos **Unplace Component (**Figura 34**)**.

Cuando se quita un componente se mueve a la lista de de Unplaced Components, volvemos a seleccionarlo en esta lista y lo eliminamos.

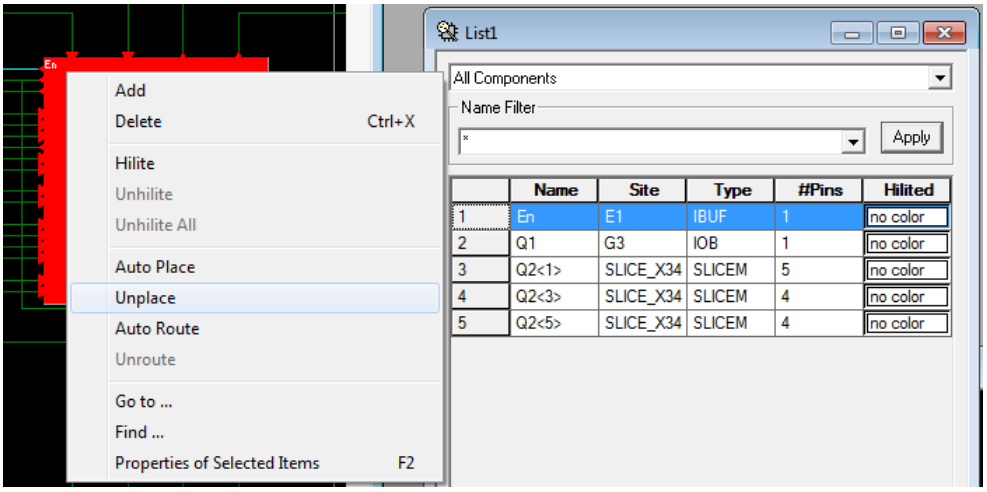

Figura 34: FPGA Editor. Quitar componente

Para utilizar la Hard Macro, hay declarar sus pines externos. Para ello hay que seleccionar los pines que pertenecen a las señales de entradas y salidas e ir a **Edit/Add Hard Macro External Pin**. Es muy importante que el nombre de los pines sea exactamente igual al de las entradas y salidas declaradas en el código VHDL (Figura 35).

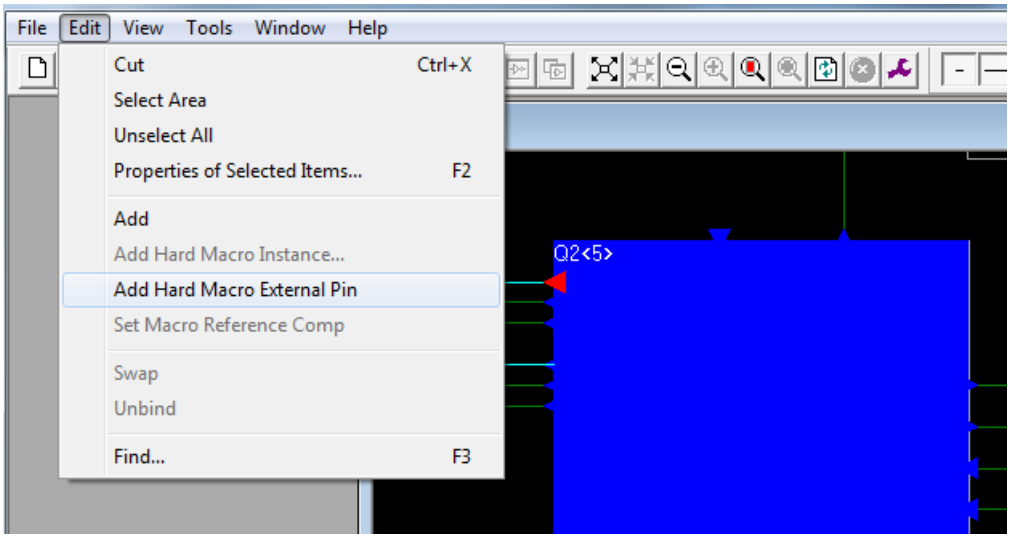

Figura 35: FPGA Editor. Añadir pin externo de la Hard Macro.

Finalmente, hay que seleccionar el componente de la Hard macro como su referencia. Para ello se selecciona el componente e ir a **Edit/Set Macro Reference Component**.

Para añadir la Hard Macro a un proyecto, hay que mover el archivo .nmc generado al directorio del proyecto e instanciarla como cualquier otro componente.

Bibliografía

[1] Maes R. *Physically unclonable functions : Constructions, properties and applications.* 1st ed. Berlin, Heidelberg: Springer; 2013. 10.1007/978-3-642-41395-7.

[2] Ravikanth PS. *Physical one -way functions.* ProQuest Dissertations Publishing; 2001.

[3] ChipScope Pro Debugging Overview. https://www.xilinx.com/itp/xilinx10/isehelp/ise_c_process_analyze_design_using_chipscope.htm

[4] Spartan-3E FPGA Starter Kit Board User Guide. https://www.xilinx.com/itp/xilinx10/isehelp/ise_c_process_analyze_design_using_chipscope.htm

[5] Spartan-3E FPGA Family Data Sheet. https://www.xilinx.com/support/documentation/data_sheets/ds312.pdf

[6] Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., & Wolters, R. (2006). Read-Proof Hardware from Protective Coatings. In L. Goubin & M. Matsui (Eds.), *Cryptographic Hardware and Embedded Systems - CHES 2006: 8th International Workshop, Yokohama, Japan, October 10-13, 2006. Proceedings* (pp. 369–383). Berlin, Heidelberg: Springer Berlin Heidelberg.

[7] D. E. Holcomb, W. P. Burleson and K. Fu, "Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers," in *IEEE Transactions on Computers*, vol. 58, no. 9, pp. 1198-1210, Sept. 2009. doi: 10.1109/TC.2008.212

[8] Platonov, M. *SRAM-Based Physical Unclonable Function on an Atmel ATmega Microcontroller*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2013.

[9] S. S. Kumar, J. Guajardo, R. Maes, G. J. Schrijen and P. Tuyls, "Extended abstract: The butterfly PUF protecting IP on every FPGA," *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, Anaheim, CA, 2008, pp. 67-70. doi:10.1109/HST.2008.4559053

[10] J. W. Lee, Daihyun Lim, B. Gassend, G. E. Suh, M. van Dijk and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," *2004 Symposium on VLSI Circuits. Digest of Technical Papers (IEEE Cat. No.04CH37525)*, 2004, pp. 176-179. doi: 10.1109/VLSIC.2004.1346548

[11] Daihyun Lim, J. W. Lee, B. Gassend, G. E. Suh, M. van Dijk and S. Devadas, "Extracting secret keys from integrated circuits," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 13, no. 10, pp. 1200-1205, Oct. 2005. doi: 10.1109/TVLSI.2005.859470

[12] Vika, R. (2011). F Aculty of S Cience and T Echnology Master ' S Thesis, 2016.

[13] Gassend, B. *Physical Random Functions*. Dissertation thesis. Massachusetts Institute of Technology, 2003. http://www.textfiles.com/bitsavers/pdf/mit/lcs/tr/MIT-LCS-TR-881.pdf

[14] Suh, G. E., & Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. *Proceedings of the 44th Annual Conference on Design Automation - DAC '07*, 9. https://doi.org/10.1145/1278480.1278484

[15] C. E. D. Yin and G. Qu, "LISA: Maximizing RO PUF's secret extraction," *2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, Anaheim, CA, 2010, pp. 100-105. doi: 10.1109/HST.2010.5513105

[16] Maiti, A., & Schaumont, P. (2009). Improving the quality of a Physical Unclonable Function using configurable Ring Oscillators. *2009 International Conference on Field Programmable Logic and Applications*, 703–707.

[17] Xin, X., Kaps, J. P., & Gaj, K. (2011). A configurable ring-oscillator-based PUF for Xilinx FPGAs. *Proceedings - 2011 14th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, DSD 2011*, 651–657.

[18] Maes, R., & Verbauwhede, I. (2010). Physically Unclonable Functions: A Study on the State of the Art and Future Research Directions. *Towards Hardware-Intrinsic Security*, (71369), 3–37. https://doi.org/10.1007/978-3-642-14452-3

[19] Wood G. Costly counterfeit electronic components in the supply chain can also be a safety concern | IHS blogs. http://blog.ihs.com/costly-counterfeit-electronic-components-in-the-supply-chain-can-also-be-a-safety-concern. Updated 2016. Accessed Jun 21, 2017.